

AD-A058 406

COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK). VOLU--ETC(U)
APR 78 D J SANDERS, P F MAYKRANTZ, J M HERRON

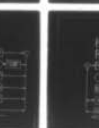
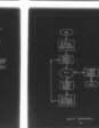
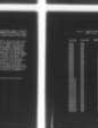
F/G 15/7

UNCLASSIFIED

CCTC-CSM-MM-9-77-VOL-4-PT SBIE-AD-E100 085

NL

1 OF 5
ADA
058406



AD No. _____

DDC FILE COPY

ADA 058406

**C
C
T
C**

**DEFENSE
COMMUNICATIONS
AGENCY**

THIS DOCUMENT HAS BEEN
APPROVED FOR PUBLIC
RELEASE AND SALE; ITS
DISTRIBUTION IS UNLIMITED.

(12)

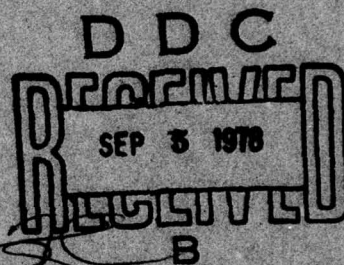


**COMMAND
& CONTROL
TECHNICAL
CENTER**

E100 085
**COMPUTER SYSTEM MANUAL
CSM MM 8-77
VOLUME IV
PART II
15 APRIL 1978**

LEVEL III

AD58436



**THE CCTC QUICK-REACTING
WAR GAMING SYSTEM (QUICK)**

**VOLUME IV
SORTIE GENERATION SUBSYSTEM PART II**

PROGRAM MAINTENANCE MANUAL

70 00 22 001

COMMAND AND CONTROL TECHNICAL CENTER

9 CCTC-
Computer System Manual, CSM-MM-9-77-VOL-4-pt-2

11 15 April 1978

6 THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK).
Volume IV, Sortie Generation Subsystem,
Part II,
Program Maintenance Manual.

12 460p.

10 Dale J./Sanders,
Paul F. M./Maykrantz
Jim M./Herron

DDC
RECEIVED
SEP 5 1978
B

18 SBIE

SUBMITTED BY:

19 AD-E100 085

C. G. Thompson
C. G. THOMPSON
Project Officer

APPROVED BY:

Fredric A. Graf
FREDERIC A. GRAF, JR.
Captain, U. S. Navy
Deputy Director, NMCS ADP

Copies of this document may be obtained from the Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314.

Approved for public release; distribution unlimited.

409658

LB

ACKNOWLEDGMENT

This document was prepared under the direction of the Chief for Military Studies and Analyses, CCTC, in response to a requirement of the Studies, Analysis, and Gaming Agency, Organization of the Joint Chiefs of Staff. Technical support was provided by System Sciences, Incorporated under Contract Number DCA100-75-C-0019.

ACCESSION TO		
NTIS	NTIS Section	<input checked="" type="checkbox"/>
DDC	DDC Section	<input type="checkbox"/>
UNANNOUNCED		<input type="checkbox"/>
JUSTIFICATION		
BY		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	AVAIL.	and/or SPECIAL
A		

CONTENTS

Part I

Section	Page
ACKNOWLEDGMENT	ii
ABSTRACT.....	viii
1. GENERAL.....	1
2. FOOTPRNT MODULE.....	5
3. POSTALOC MODULE.....	111

Part II

4. PLANOUT MODULE.....	261
4.1 General Purpose.....	261
4.1.1 Sortie Completion.....	261
4.1.2 Sortie Change.....	261
4.1.3 External Interface.....	261
4.1.4 Modes of Execution.....	262
4.2 Input.....	262
4.2.1 RECALC Mode Input.....	262
4.2.2 NonRECALC Mode Input.....	262
4.3 Output.....	263
4.3.1 Sortie Completion Output.....	263
4.3.2 Sortie Change Output.....	263
4.3.3 External Interface Output.....	263
4.4 Concept of Operation.....	267
4.5 Identification of Subroutine Functions.....	267
4.5.1 Sortie Completion Function.....	267
4.5.2 Sortie Change Function.....	267
4.5.3 External Interface Function.....	271
4.6 Common Blocks.....	271
4.7 Subroutine ENTMOD.....	294
4.7.1 Subroutine CLINDATA.....	308
4.7.2 Subroutine CONVLL.....	310
4.7.3 Subroutine GEOGET.....	312
4.7.4 Subroutine SNAPCON.....	319
4.7.5 Subroutine WEPDATA.....	324
4.7.6 Function XLL.....	332
4.8 Subroutine PLNTPLAN.....	335
4.8.1 Subroutine ALTPLAN.....	343
4.8.2 Subroutine ADJUST.....	377
4.8.3 Subroutine CHGTIM.....	393

Section	Page
4.8.4 Subroutine DECOYADD.....	396
4.8.5 Subroutine DISTIME.....	406
4.8.6 Subroutine FINDME.....	410
4.8.7 Subroutine FLTSORT.....	416
4.8.8 Subroutine FLYPOINT.....	424
4.8.9 Subroutine INITANK.....	426
4.8.10 Subroutine KERPLUNK.....	428
4.8.11 Subroutine LAUNCH.....	432
4.8.12 Subroutine LNCHDATA.....	439
4.8.13 Subroutine PLAN.....	449
4.8.14 Subroutine PLANBOMB.....	487
4.8.15 Subroutine PLANTMIS.....	499
4.8.16 Subroutine POST.....	504
4.8.17 Subroutine POSTLAUN.....	506
4.8.18 Subroutine SNAPIT.....	508
4.8.19 Subroutine SNAPOUT.....	510
4.8.20 Subroutine SORBOMB.....	513
4.8.21 Subroutine SWTCHALT.....	516
4.9 Subroutine PLANTANK.....	518
4.9.1 Subroutine PRNTAB.....	525
4.9.2 Subroutine VAM.....	527
4.10 Subroutine INTRFACE.....	539
4.10.1 Subroutine ABOUT.....	545
4.10.2 Subroutine FINDTIME.....	550
4.10.3 Function IAZIM.....	552
4.10.4 Subroutine IFSET.....	554
4.10.5 Function IFUNCT.....	559
4.10.6 Subroutine INFORM.....	561
4.10.7 Function NOP.....	564
4.10.8 Subroutine PRNTOFFS.....	566
4.10.9 Subroutine RDCLAUSE.....	568
4.10.10 Subroutine STOUT.....	574
4.10.11 Subroutine TYPFIND.....	578
4.10.12 Subroutine XSET.....	580
REFERENCES.....	589
APPENDIXES	
A. Sortie Generation Algorithms and Concept.....	591
B. An Algorithm for the Traveling Salesman Problem.....	641
DISTRIBUTION.....	657
DD Form 1473.....	659

ILLUSTRATIONS (PART II)

Number		Page
59	STRIKE Tape Format.....	266
60	STRIKE Format (A and B Cards).....	268
61	PLANOUT Module Macro Flow.....	270
62	Subroutine ENTMOD.....	295
63	Subroutine CLINDATA.....	309
64	Subroutine CONVLL.....	311
65	Subroutine GEOGET.....	313
66	Subroutine SNAPCON	321
67	Subroutine WEPDATA	325
68	Function XLL	333
69	Subroutine PLNTPLAN	336
70	Subroutine ALTPLAN	347
71	High-Altitude Adjustment	380
72	Low-Altitude Adjustment	380
73	Increase In Low-Altitude Flight	381
74	Subroutine ADJUST	384
75	Subroutine CHGTIM	394
76	Subroutine DECOYADD	399
77	Subroutine DISTIME	407
78	Subroutine FINDME	412
79	Subroutine FLT SORT	417
80	Subroutine FLYPOINT.....	425
81	Subroutine INITANK	427
82	Subroutine KERPLUNK.....	429
83	Determination of ASM Aim Point	433
84	LAUNCH Procedure Outline	435
85	Computation of Flight Path Aim Point	436
86	Subroutine LAUNCH	438
87	Subroutine LNCHDATA	440
88	Subroutine PLAN (Macro Flowchart)	450
89	Subroutine PLAN -- Block 20: Determine Type of Plan ...	452
90	Subroutine PLAN -- Block 24: Initialize Plan	454
91	Subroutine PLAN -- Block 25: Post Launch Event.....	455
92	Subroutine PLAN -- Block 26: Post Refuel Events.....	457
93	Acceptable Locations for Refuel Area (Shaded Section)...	462
94	Subroutine PLAN -- Block 27: Initialize Plan With Respect to GOLOW Range.....	464
95	Subroutine PLAN -- Block 30: Process Precorridor Legs and Apply GOLOW1.....	465
96	Example of Precorridor Legs.....	468
97	Subroutine PLAN -- Block 31: Post Corridor Events	470
98	Subroutine PLAN -- Block 40: Adjust /OUTSRT/ for ASM Events.....	474
99	Illustration of ASM Event Adjustment.....	479

Number		Page
100	Subroutine PLAN -- Block 50: Apply GOLOW2 Before First Target.....	480
101	Subroutine PLAN -- Block 60: Post Depenetration Events.	486
102	Path of Typical Bomber Sortie.....	488
103	Subroutine PLANBOMB.....	490
104	Subroutine PLANTMIS.....	501
105	Subroutine POST.....	505
106	Subroutine POSTLAUN.....	507
107	Subroutine NAPIT.....	509
108	Subroutine SNAPOUT.....	511
109	Subroutine SORBOMB.....	514
110	Subroutine SWTCHALT.....	517
111	Subroutine PLANTANK.....	521
112	Subroutine PRNTAB.....	526
113	Base/Refuel Area Sample Matrix.....	528
114	Subroutine VAM.....	531
115	Subroutine INTRFACE.....	540
116	Subroutine ABOUT.....	546
117	Subroutine FINDTIME.....	551
118	Function IAZIM.....	553
119	Subroutine IFSET.....	555
120	Function IFUNCT.....	560
121	Subroutine INFORM.....	562
122	Function NOP.....	565
123	Subroutine PRNTOFFS.....	567
124	Subroutine RDCLAUSE.....	569
125	Subroutine STOUT.....	575
126	Subroutine TYPFIND.....	579
127	Subroutine XSET.....	581
128	Illustrative Curvilinear Functions.....	595
129	Exemplar Configuration of Missiles in a Group.....	607
130	Typical Bomber Flight Route.....	615
131	High-Altitude and Adjustment	619
132	Increase in Low-Altitude Flight.....	621
133	Illustration of ASM Launch Point Calculation.....	622
134	Assigning a Refuel Area (Automatic).....	626
135	Formulation of a Tanker Allocation Problem.....	633
136	Coordinate System for Missile Timing Calculations.....	636
137	Relation of R_{1j} to Great Circle Plane.....	637
138	Diagram of T Vector.....	639
139	Cost Matrix for a 6-City Problem.....	645
140	Cost Matrix after Reducing Rows and Columns.....	646
141	Start of Tree.....	646
142	Flow Chart of the Algorithm.....	648
143	Matrix After Deletion of Row 1 and Column 4, and First Branching.....	650
144	Final Tree.....	652

TABLES (PART II)

Number		Page
4	Attributes in Sortie Event Records.....	264
5	Bomber Events Recognized by PLNTPLAN.....	265
6	STRIKE and ABTAPE Fields in Block /DEFVAR/.....	272
7	PLANOUT Module Internal Common Blocks.....	274
8	Possible Values of a and b.....	320
9	Launch Priority.....	397
10	List of Admissible Input Events by Type and Information Relevant to Each.....	456
11	Tanker Plan.....	520
12	List of Information Required by PLANOUT.....	613
13	Launch Priority.....	617
14	Scaled Height of Burst Selection.....	630
15	Mean and Standard Deviation of T for Random Distance Matrices.....	656

ABSTRACT

The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide output summaries, and produce input tapes to simulator subsystems external to QUICK. QUICK has been programmed in FORTRAN for use on the CCTC HIS 6000 computer system.

The QUICK Program Maintenance Manual consists of four volumes: Volume I, Data Management Subsystem; Volume II, Weapon/Target Identification Subsystem; Volume III, Weapon Allocation Subsystem; Volume IV, Sortie Generation Subsystem. The Program Maintenance Manual complements the other QUICK Computer System Manuals to facilitate maintenance of the war gaming system. This volume, Volume IV, is in two parts providing the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the modules and subroutines of the Sortie Generation subsystem. Companion documents are:

a. **USERS MANUAL**

Users Manual CSM UM 9-77, Volume I

Users Manual CSM UM 9-77, Volume II

Users Manual CSM UM 9-77, Volume III

Users Manual CSM UM 9-77, Volume IV

Provides detailed instructions for applications of the system.

b. **TECHNICAL MEMORANDUM**

Technical Memorandum TM 153-77

Provides a nontechnical description of the system for senior management personnel.

SECTION 4. PLANOUT MODULE

4.1 General Purpose

PLANOUT accepts skeletal bomber sorties from module POSTALOC and adds serial bomber events (i.e., doglegs) to each sortie. PLANOUT, also, accepts missile sorties developed by module FOOTPRNT. For both type systems, PLANOUT ultimately generates output tapes as input to systems external to the QUICK system.

In addition to the normal development of output tapes, PLANOUT permits user intervention in altering sortie development in two fashions. First, the user has commands that permit changes to the data base stored sorties without requiring a new allocation. Second, capability is provided to postprocess the output tapes record(s); not data base contents. The postprocessing commands are completely generalized whereby any item within a record may be altered.

Accordingly the functions of the PLANOUT Module fall into three categories: Sortie Completion, Sortie Change and External Interface.

4.1.1 Sortie Completion. Bomber and missile sorties are processed and tanker sorties are created. Among the processing functions performed are:

- o Assigning refuel areas to bombers and allocating tankers to service them.
- o Calculating ASM launch points
- o Determining where change altitude and launch decoy events should occur
- o Coordinating bomber and missile launch times according to user parameters
- o Calculating distances and time between all events of each plan.

4.1.2 Sortie Change. This function allows the user to make minor changes to sorties which do not require a new allocation. The user is given control, through sortie change clauses, over targets assigned to a specific sortie, the weapon offsets, time of delivery, and height of burst. The user may also create complete non-MIRV missile sorties. These requests alter items stored within the data base.

4.1.3 External Interface. This function allows the user to specify the production of two output tapes: a STRIKE tape which is subsequently used as input to the external damage assessment system SIDAC, and Sortie Specifications tape (ABTAPE) which provides an interface with the NEMO and ESP simulators. The STRIKE tape contains a set of data for

each weapon delivery reflected in the generated sorties. The ABTAPE contains two types of information for each missile or bomber mission: the first is general descriptive data for the mission; and the second is data reflecting the ordered sequence of flight route legs for the mission.

The user also has the option of altering the data on these files by specifying the data to be altered (SETTING clause) and the conditions under which this alteration is to occur (IF clause).

The logical output unit number for these tapes is preset within the module as follows: STRIKE tape = LTN4, ABTAPE = LTN16.

4.1.4 Modes of Execution. Traditionally PLANOUT initially performs the Sortie Completion function and upon user examination of this output separate executions follow whereby the user alters the initial allocation; either data base items or record(s) on tapes for use as input to systems external to QUICK. The user controls the mode of execution.

4.2 Input

Data base definition to the PLANOUT module depends upon two modes of operation available to the user. These modes, identified by adverbs, are: RECALC and nonRECALC.

4.2.1 RECALC Mode Input. This is the mode in which the user exercises the Sortie Completion function, though it is not restricted to this function. The normal circumstance under which this mode would be used would be immediately after execution of the POSTALOC and FOOTPRNT modules. The user may wish to use the RECALC mode at a later date (that is, execute the Sortie Completion function more than once during the same study), but it must be used at least once before either the Sortie Change or External Interface functions are exercised. The text English input required for proper execution under this mode -- chiefly characterized by the presence of the RECALC adverb -- is fully described in User's Manual UM 9-77, Volume IV.

4.2.2 NonRECALC Mode Input. In this mode the user may exercise either the sortie change or external interface function or both. The major precondition of the data base is that the PLANOUT module has to have been executed at least once previously in the RECALC mode. The text English input required -- characterized by the absence of the RECALC adverb -- is fully described in User's Manual UM 9-77, Volume IV.

4.3 Output

4.3.1 Sortie Completion Output. The sortie completion function is basically designed to accept sorties prepared by the POSTALOC and FOOTPRNT modules. In the case of missile sorties (from FOOTPRNT), the output consists of updating the sortie table (SRTYTB, record type 33) with the launch time (attribute SLOW1 is used). In the case of bomber sorties (from POSTALOC) the output consists of updates to existing sortie events records (SRTEVA, record type 50 and SRTEVB, record type 53), and additional SRTEVB records to complete the sortie. The attributes contained in these record types (the SRTEVA record is distinct from the SRTEVB record in that it represents a weapon assignment and, as such, is linked to an ASSIGN record-record type 70) appear in table 4. Two of the values, "type of event" and "place index" are further explained in table 5. The event types which are added as a part of the sortie completion function are:

- o Launch
- o Refueling
- o ASM launches
- o Flight altitude changes

After all missile and bomber sorties have been processed and completed, tanker sorties are created including sortie table (SRTYTB) and sortie events (SRTEVB).

4.3.2 Sortie Change Output. The sortie change function makes alterations to existing sortie events. It is principally designed to effect those events produced by POSTALOC or FOOTPRNT. After all changes to a particular sortie have been made, the processing is much the same as that for the sortie completion function (i.e., the "changed" sorties are now "completed"), and as such so is the output. Certain sortie changes may cause the addition, deletion and/or modification of sortie table (SRTYTB), sortie events (SRTEVA or SRTEVB) or weapon assignments (ASSIGN). The ACARD clause (see UM 9-77, Volume IV) will create one each of: a sortie table, a sortie event (SRTEVA) and a weapon assignment.

4.3.3 External Interface Output. This function produces, at user direction, two types of tapes of differing formats for use in external processors.

- a. The STRIKE tape is produced on logical tape number (LTN) 4 for use with the single Integrated Damage Analysis Capability System (SIDAC). It contains a "strike" card for each weapon assignment from either missile or bomber systems. The format appears in figure 59.
- b. The sortie specifications tape (ABTAPE), is produced on LTN 16 for use in the Event Sequential Program (ESP) and the Nuclear

Table 4. Attributes in Sortie Event Records

<u>Attributes</u>	<u>Description</u>
LAT	Latitude of event
LONG	Longitude of event
SLOCATTR	Local attrition *
SCUMSURV	Cumulative survival probability
SDELTIME	Time since previous event
SDAMEXP	Expected damage achieved *
SEVCODE	Type of event **
SPLACE	Place index **
SCHANGE	Blank unless event was inserted or altered by the sortie change function in which case its value is "Cmm/dd/" where "mm" is the month and "dd" is the day on which the change was made

* Applies only to weapon assignment event (SRTEVA)

** See table 5.

Table 5. Bomber Events Recognized by PLNTPLAN

<u>Type of Event</u>	<u>Event Type</u>	<u>Place Index</u>	<u>Event Names Used In</u>	
			<u>/EVENTS/</u>	<u>OUTPUT PRINT</u>
Launch	2	Base Index	LAUNB	LAUNCH B
REFUEL	4	Refuel Index	LREFUEL	REFUEL
Local Attrition or Drop Bomb	8	Target Index	LOCLATTR	DROPBOMB
Launch ASM	14	ASM Type	LAUNASM	LAUN ASM
ASM Target	--	Target Index	--	ASM TGT
Launch Decoy	15	*	LAUNDCOY	LAUNDCOY
Change Altitude	17	1 for Go High, 0 for Go Low	LOHI	CHANGALT
Recover	16	Recovery Base Index	LANDHO	RECOVER
Abort	13	*	LABORT	ABORT
Enter Refuel	11	Refuel Area Index	LENTEREF	ENTERREF
Leave Refuel	12	*	LEAVEREF	LEAVERED
Go High	18	*	IGOHI	GO HIGH
Go Low	19	*	IGOLOW	GO LOW
Dogleg	20	Penetration Corridor Index	LEGDOG	DOGLEG

* Place index not applicable.

<u>FIELD</u>	<u>CARD COLUMNS</u>	<u>LABEL</u>	<u>RANGE</u>	<u>DESCRIPTION</u>
1	1		S	STRIKE Card indicator
2	2		0	Constant
3	3		1-9	Command/function code
4	4-8	SSSN	1-99999	Sortie sequence number
5	9-10	SDTM	01-12	Month
	11-12		01-31	Day
	13-14		00-23	Hour
	15-16		00-59	Minutes
6	17-18		00-59	Seconds
				Of weapon detonation
7	19-24	SLAT	DDMMSS where	DD = degrees
				MM = minutes
				SS = seconds
				Latitude of desired ground zero (DGZ)
	25		N or S	North or South
8	26-32	SLON	DDMMSS where	DDD = degrees
				MM = minutes
				SS = seconds
				Longitude of DGZ
	33		E or W	East or West
9	34-38	SDES	2 Alpha, 3 Numeric	Target designator code
10	39-40	SPLS	-1-99	PLS-Probability* of pre-launch survival
11	41-42	SPTP	-1-99	PTP-Penetration probability*
12	43-44	SWSR	-1-99	WSR-Weapon system reliability*
13	45	SREG	1-9	Region code
14	46-48	SFYR	000-999	Fission/yield ratio
15	49			Blank
16	50-54	SYLD	00001-99999	Yield
17	55-57	SHOB	000-999	Height of burst (hundreds of feet)
18				
19	58-60	SCEP	000-999	CEP in 100s of feet
20	61-62	STSK	2 Alpha	Task/subtask code
21	63-64	SCLO	2 Alpha	Country code for target location
22	65-66	SCOW	2 Alpha	Country code for target owner
23	67-68	SPAT	00-99	Attrition probability* (i.e., percent change of attrition)
24	69	SSEQ	0-9	Sequential Warhead Number when operation code is 7, 10, or 11. Otherwise, blank
25	70-71	SPTC	01-99	Plane type code
26	72-73	SWTC	01-99	Weapon type code
27	74-77	SUNT	0001-9999	Unit number (INDEXNO of launch base)
28	78-79		00-99	Sortie number
29	80			Blank

* A printed probability of -1 implies a value of 100 (necessary since only two digits are reserved for probability representation).

Figure 59. STRIKE Tape Format

Exchange Model (NEMO). This tape contains a set of BCD card images for each missile, bomber, and tanker sortie. A card set consists of one "A" card which contains general descriptive information and a variable number of "B" cards which define the individual events of the sortie. The card formats are described in figure 60.

After an initial execution of the external interface function, the user may desire to alter the fields output on these two tapes. This is done via the IF and SETTING clauses detailed in UM 9-77, Volume IV. The data base sorties themselves are not changed by these clauses only the fields in the output tapes. Any of the fields of any subset of the record types for either output tape may be altered as the tape is being created via these clauses.

4.4 Concept of Operation

The main subroutine (ENTMOD) scans the input clauses setting values in the /ADVRB/ common block. Through the switches RECSW, STRSW, ABSW and CHNGSW, the remainder of the module flow is controlled. Referring to figure 61, first the RECALC mode switch (RECSW) is checked and if the mode was selected, PLNTPLAN is executed to perform the sortie completion function (except tanker sortie creation). Next, the CHNGSW switch (set if any ACARD, CCARD or ICARD clauses are input) is checked and if true, ALTPLAN is called to perform the sortie change function. Then, if RECALC mode was selected, PLANTANK is called for tanker sorties. Finally, if either ABSW or STRSW are true (indicating the presence of the ABTAPE or STRIKE adverb respectively), INTRFACE is called to perform the external interface function.

4.5 Identification of Subroutine Functions

4.5.1 Sortie Completion Function. The driving subroutine of this function is PLNTPLAN. This subroutine reads in the missile and bomber sorties as they exist in the integrated data base as created by FOOTPRINT and POSTALOC and, perhaps, as modified by previous executions of PLANOUT. These sorties are finalized one at a time. For bomber sorties, the PLANBOMB subroutine is called which controls the addition of sortie events to the bomber plan. Among the processing functions performed on the input phases are: assigning refuel areas; calculating ASM launch points; determining where altitude changes and decoy launch points should occur; and coordinating launch times according to user parameters.

Missile sortie processing takes place in subroutine PLANTMIS which assigns launch times based on user supplied coordination parameters.

In an overlay separate from PLNTPLAN, PLANTANK generates tanker plans such that all bombers will be serviced as required.

4.5.2 Sortie Change Function. The driving subroutine of this function is ALTPLAN. This subroutine scans user request clauses to accumulate

SORTIE SPECIFICATION: "A" CARD FORMAT

<u>FIELD</u>	<u>CARD COLUMNS</u>	<u>LABEL</u>	<u>RANGE</u>	<u>DESCRIPTION</u>
1	1		A	A-card indicator
2	2-4	ANUM	001-999	A-card number
3	5-8	AUNT	0001-9999	Unit number
4	9-10	ASNO	01-99	Sortie number
5	13-14	APTC	01-99	Plane type code
6	15		0	
7	16-17			Blank
8	18		0	
9	19-22	AREF	0000-9999	Reference time (launch time in hours and minutes)
10	23		1	Time reference (1 = launch)
11	24-30		0000000	
12	31			Blank
13	32-34	APTA	N/A	Plane type alpha-numeric code
14	35			Blank
15	36-37	ALCC	N/A	Country code of launch base
16	38			Blank
17	39-40	AFUN	1-9	SAGA Vehicle-Function code 1 = ICBM 2 = IREB 3 = MREB 5 = SSB/SSBN 6 = SSGN 7 = LRA 0, 4, 8, 9 not used
18	41-80			Blank
19	N/A			Card identification number (added to print of card image not contained on tape)

SORTIE SPECIFICATION: "B" CARD FORMAT

<u>FIELD</u>	<u>CARD COLUMNS</u>	<u>LABEL</u>	<u>RANGE</u>	<u>DESCRIPTION</u>
1	1		B	B-card indicator
2	2-4	BNUM	001-999	B-card number
3	5-8	BUNT	0001-9999	Unit number
4	9-10	BSNO	01-99	Sortie number
5	11-12	BFLN	01-99	Flight leg number
6	13-14	BEUT	01-14	Event or operation type indicator
			1	Takeoff
			2	Aerial refueling
			3-4	Dogleg
			6	ASM launch

Figure 60. STRIKE Format (A and B Cards)
(Part 1 of 2)

<u>FIELD</u>	<u>CARD COLUMNS</u>	<u>LABEL</u>	<u>RANGE</u>	<u>DESCRIPTION</u>
			7	ASM on target
			8	Decoy release
			9	Decoy impact
			10	Missile or bomb on target
			11	MIRV or target
			13	Recovery if bomber; splash if air breathing missile
			14	Splash (ballistic missiles)
7	15-19	BLOC		Location identifier for given operation 1 = Base index 2 = Area number 6 = "1" 7 = Target DESIG code 8 or 9 = "1" 10-11 = Target DESIG code 13 = Recovery base INDEXNO if bomber
8	20-25	BLAT	DDMMSS	Latitude at end of leg is degrees, minutes, and seconds
9	26-33	BLON	DDDDMMSSX	Longitude at end of leg is degrees, minutes, and seconds; East or West
10	34	BMOD		Mode of operation 1 High altitude 4 Low altitude
11	35		0	Time of event in hours, minutes, and seconds
12	36-41	BTIM	HHMMSS	
13	42		S	Southern latitude indicator (if latitude is North, column 42 is blank)
14	43-44	BSEQ	01-90	Sequential index within unit number
15	45-46			Blank
16	47-49	BAZI	0-360	Launch/Back azimuth in degrees
17	50	BECM		ECM status 0 Off 1 On
18	51		0	
19	52-53	BWAR	00-99	Warhead type
20	54	BCRA		Height of burst 0 = ground 1 = air
21	55-56		0	
22	57-58	BPTC	01-90	Plane type code
23	59-60	BTCC	2 Alpha	Code for country of target location
24	61	BRFC	1-9	Region code
25	62			Blank
26	63-64	BTSK	2 Alpha	Target task code
27	65-67	BHOB	000-999	Height of burst (hundreds of feet)
28	68-72	BYLD	00001-99999	Yield (KT)
29	73-75	BCEP	000-999	CEP (100's feet)
30	76-77		2 Alpha	Code for country target owner
31	N/A	BCOW		Card number for whole SORTIE SPECIFICATIONS tape (contained only in a print of the tape)

Figure 60. (Part 2 of 2)

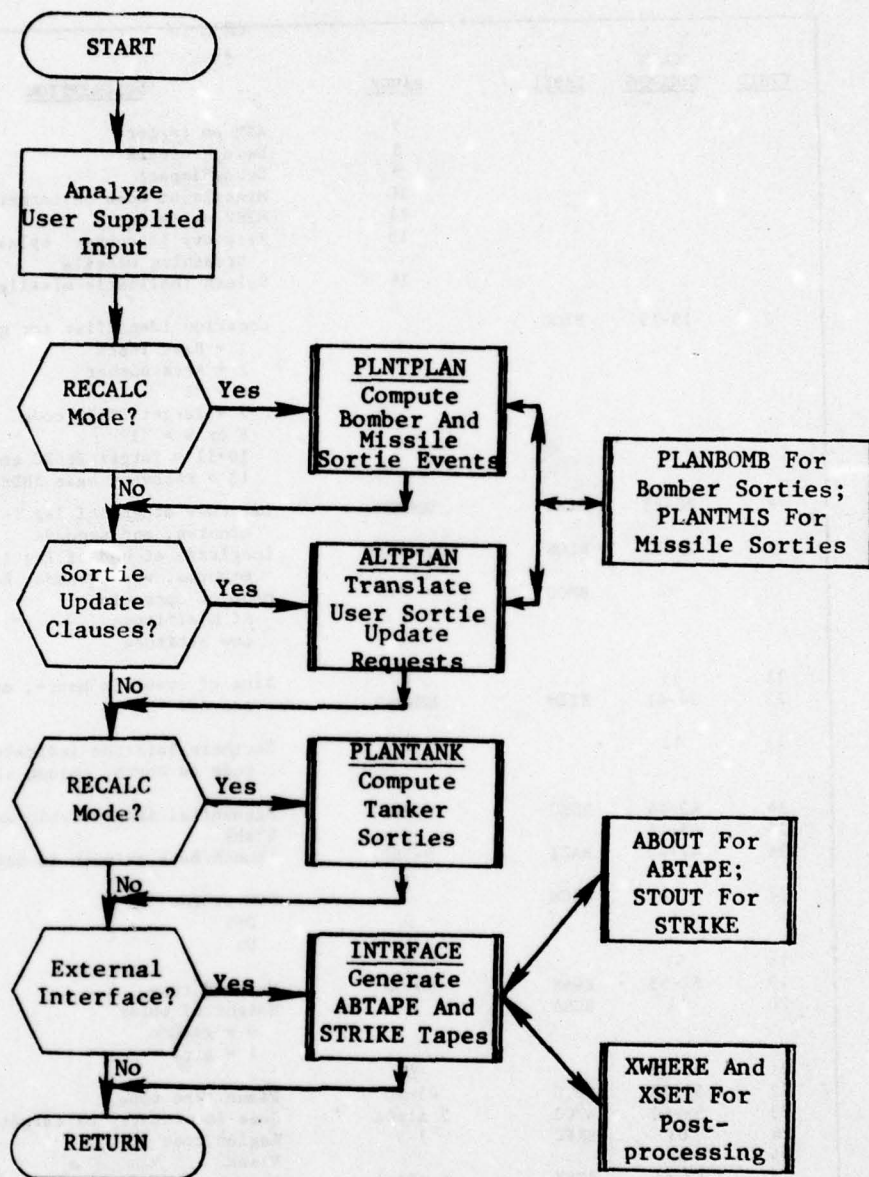


Figure 61. PLANOUT Module Macro Flow

all changes pertaining to a sortie. Then each change request is carried out and the appropriate plan processor (PLANBOMB or PLANTMIS) is called to recomplete the sortie. Any new sorties (resulting from ACARD clauses) are inserted before the first tanker sortie and the numbers of the tanker sorties changed to reflect the additions.

4.5.3 External Interface Function. INTRFACE is the driver of this function. Sorties are read one at a time and the two tapes produced. Subroutine ABOUT is called for A or B cards for the ABTAPE and subroutine STOUT is called to produce STRIKE tape records. Both of these routines have the same procedural design. First the appropriate information is extracted from the sortie data and stored in common block /DEFVAR/ in array positions which correspond with the output tape fields, as shown in figures 59 and 60. The precise correspondences of the /DEFVAR/ block to these fields appears in table 6. After the data is stored in /DEFVAR/, any postprocessing desired for the tape in question is performed. The IF and SETTING clause pairs pertaining to the output tape being produced are executed in the input order. For each pair, the IF clause is first executed using the XWHERE utility subroutine. This subroutine will return a switch stating whether the current state of the data base (/DEFVAR/ in particular) satisfies the conditions stated in the IF clause. If the conditions are satisfied the corresponding SETTING clause is carried out by subroutine XSET. When all postprocessing has been completed, the data items are converted to BCD format via the INFORM subroutines and written onto tape.

4.6 Common Blocks

The common blocks internal to the PLANOUT module are shown in table 7.

Table 6. STRIKE and ABTAPE Fields In Block /DEFVAR/
(Part 1 of 2)

<u>Tape Name</u>	<u>Field Name</u>	<u>/DEFVAR/Index</u>
STRIKE	SSSN	1
STRIKE	SDTM	2
STRIKE	SLAT	3
STRIKE	SLON	4
STRIKE	SDES	5
STRIKE	SPLS	6
STRIKE	SPTP	7
STROKE	SWSR	8
STRIKE	SREG	9
STRIKE	SFYR	10
STRIKE	SYLD	11
STRIKE	SHOB	12
STRIKE	SCEP	13
STRIKE	STSK	14
STRIKE	SCLO	15
STRIKE	SCOW	16
STRIKE	SPAT	17
STRIKE	SSEQ	18
STRIKE	SPTC	19
STRIKE	SWTC	20
STRIKE	SUNT	21
ABTAPE(A)	ANUM	22
ABTAPE(A)	AUNT	23
ABTAPE(A)	ASNO	24
ABTAPE(A)	APTC	25
ABTAPE(A)	AREF	26
ABTAPE(A)	APTA	27
ABTAPE(A)	ALCC	28
ABTAPE(A)	AFUN	29
ABTAPE(B)	BNUM	30
ABTAPE(B)	BUNT	31
ABTAPE(B)	BSNO	32
ABTAPE(B)	BFLN	33
ABTAPE(B)	BEUT	34
ABTAPE(B)	BLOC	35
ABTAPE(B)	BLAT	36
ABTAPE(B)	BLON	37
ABTAPE(B)	BMOD	38
ABTAPE(B)	BTIM	39
ABTAPE(B)	BSEQ	40
ABTAPE(B)	BAZI	41

Table 6. (Part 2 of 2)

<u>Tape Name</u>	<u>Field Name</u>	<u>/DEFVAR/Index</u>
ABTAPE(B)	BECM	42
ABTAPE(B)	BWAR	43
ABTAPE(B)	BCRA	44
ABTAPE(B)	BPTC	45
ABTAPE(B)	BTCC	46
ABTAPE(B)	BRFC	47
ABTAPE(B)	BTSK	48
ABTAPE(B)	BHOB	49
ABTAPE(B)	BYLD	50
ABTAPE(B)	BCEP	51
ABTAPE(B)	BCOW	52

Table 7. PLANOUT Module Internal Common Blocks
(Part 1 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
ADVRB		The contents of this block are an input clause summary. The variables are either switches which indicate the presence of certain clauses, the "index" or ordinal of the adverb in the "verb adverb array" in the input tables (see MM 9-77, Volume I), a count of a certain type of adverb, or the "pointer" to the beginning of the clause in the input.
	STRSW	True if STRIKE tape is desired
	ABSW	True if ABTAPE is desired
	CHNGSW	True if Sortie Change function is to be used
	ISTRA	Index of first STRIKE SETTING clause
	LSTRA	Index of last STRIKE IF clause
	IABA	Index of first ABTAPE SETTING clause
	LABA	Index of last ABTAPE IF clause
	IFCHNG	index of first sortie change clause
	LCHNG	Index of last sortie change clause
	IMIST	Index of first MISTME clause
	NMIST	Number of MISTME clauses
	IMCOR	Index of first MSLCOR clause
	NMCOR	Number of MSLCOR clause
	IGTIME	Pointer to GAMETIME clause
	IFUNCO	Pointer to FUNCOM clause
	IONPR	Pointer to ONPRINTS clause

Table 7. (Part 2 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
ARTIME	ARTIME(50)	Earliest bomber arrival time at re-fuel area I
	NBUDREF	Number of "buddy" refuelings required
	NBOMBREF(50)	Number of bombers assigned to refuel area I
	NTANKREF(50)	Number of tankers assigned to refuel area I
	IARVLS/ARVLS(2,1000)	ARVLS(1,I) = time of the Ith bomber refuel processed by PLNTPLAN; IARVLS(2,I) = the refuel area for that bomber refuel
ASMARRAY	ALAT(10)	Aim point latitude
	ALON(10)	Aim point longitude
	IFLY(10)	Fly point flag
	IDIS(10)	Distance from fly point to ASM target
	IORD(10)	Sort index
	JAY	Index communicated to PREFL1, PREFL2
	DIST	Distance communicated to PREFL1, PREFL2, POSTFLY
CALLSW		Each switch in this block is set to true after the indicated subroutine is executed. Its purpose is to prevent subsequent executions
	WDSW	Switch for WEPDATA
	CORSW	Switch for GEOGET
	LNDSW	Switch for LNCHDATA

Table 7. (Part 3 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
CONTROL	LSIDE	Side whose sorties are being used (1 = BLUE, 2 = RED)
CORCOUNT	IH	Points to line of /HAPPEN/ where current corridor begins
	KC	Number of lines in /HAPPEN/ describ- ing current corridor
	JH	Points to line of /HAPPEN/ where cur- rent depenetration corridor begins
	LC	Number of lines in /HAPPEN/ describ- ing depenetration corridor
CORRC1	MCORCH1	Maximum number of penetration corri- dor
	IDEFDST(30)	Total precorridor defended distance
	IMPRTD(30,3)	Order of importance of attrition per nautical mile
	ATPDST(30,3)	Average attrition per distance in the Jth corridor, Ith leg
CORRCHAR		Penetration corridor characteristics
	PCLAT(30)	Orientation point latitude
	PCLONG(30)	Orientation point longitude
	RPLAT(30)	Origin latitude
	RPLONG(30)	Origin longitude
	ENTLAT(30)	Entry latitude
	ENTLONG(30)	Entry longitude
	CRLNGTH (30)	Distance from entry to origin
	KORSTYLE(30)	Parameter to adjust mode of corridor penetration

Table 7. (Part 4 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
CORRCHAR (cont.)	ATTRCORR(30)	High-altitude attrition per nautical mile, unsupressed
	ATTRSUPF(30)	High-altitude attrition per nautical mile, suppressed
	HILOATTR(30)	Ratio low- to high-altitude attrition (less than one)
	DEFRANGE(30)	Characteristic range of corridor defense (nautical miles)
	NPRCRDEX(30)	Number of attrition sections
	DEFDISTX(30,3)	Length of attrition section
	ATTRPREX (30,3)	Attrition per nautical mile of attrition section
DECA	NDA	Number of corridors
	DELDIS(6)	Decoy coverage distance
	LPRIORITY(20)	Possible decoy launch priority
	LMHT(90)	Possible decoy launch event number
	NDCYRQ(20)	Pointer to array DELDIS
	NPSLN	Number of possible decoy launches
	NUMDCOYS	Number of decoys available
DINDATA	HDT(90)	Time -- for detailed history -- of event I
	KPL(90)	Place of event I
	JTP(90)	Event type of event I
	HLA(90)	Latitude of event I
	HLO(90)	Longitude of event I
	TZT(90)	Weapon offset latitude of event I

Table 7. (Part 5 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
DINDATA (cont.)	TZN(90)	Weapon offset longitude of event I
	PA(90)	Probability of arrival at target of event I
	MHT	Total number of lines in detailed plan
	NPL	Number of planned events
DINDT2	CMT(90)	Cumulative time of event I
	IWH(90)	Warhead type index of event I
DISTC	DISTC(20)	Distances between target events
DPENREF	DPLAT(50)	Depenetration latitude
	DPLONG(50)	Depenetration longitude
	QFLAT/RFLAT(20)	Refuel point latitude
	QFLONG/RFLONG(20)	Refuel point longitude
EVCOM	TELAPSE	Elapsed time to current event
	ICEV	Event count to current event
	EVLAT	Latitude of event
	EVLONG	Longitude of event
EVENTS	LAUNM	Missile launch code
	LAUNB	Bomber launch code
	LEREFUEL	Refuel code
	LOCLATTR	Local attrition or drop bomb event code
	LAUNASM	Launch ASM event code
	LAUNDCOY	Launch Decoy event code

Table 7. (Paet 6 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
EVENTS (cont.)	LANDHO	Recovery event code
	LOHI	Change Altitude event code
	MISSATTR	Missile attrition event code
	LEGGOG	Dogleg event code
	LABORT	Abort event code
	LENTEREF	Enter refuel area event code
	LEAVEREF	Leave refuel area event code
	IGOHI	Go to high altitude event code
	IGOLOW	Go to low altitude event code
GAMETIME	KDAY	Day of game
	KMON	Month of game
	KYEAR	Year of game
	HHR	H-Hour
GRPSTF	IPAY(250)	Weapon group payload index
	ITYPEX(250)	Weapon group type index
	IREGON(250)	Weapon group region
	GSBLX(250)	Weapon group prelaunch survival probability
	GPKNAV(250)	Single shot kill probability against naval targets for weapon group I
	IGCLS(250)	Weapons group class index
	IGLERT(250)	Weapon group alert status

Table 7. (Part 7 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
HAPPEN	KOUNT(30)	Number of /HAPPEN/ lines for penetration corridor
	IHAP(30)	Pointer to first line entry for penetration corridor
	MOUNT(50)	Number of /HAPPEN/ lines for depenetration corridor
	JHAP(50)	Pointer to first line entry for depenetration corridor
	JAPTYPE(250)	Attrition section indicates (1,2,3 enter section; 4,5,6 leave section)
	HAPLAT(250)	Latitude of corridor point
	HAPLONG(250)	Longitude of corridor point
	HAPDIST(250)	Distance from previous point
HILO	ISTOREHI	Number of events in /OUTSRT/ after which GO HIGH occurs
	ISTORELO	Number of event in /OUTSRT/ after which GO LOW occurs
	IGOLEFT	Set to 1 if GO LOW range is available after depenetration
	FACHI	Distance after event ISTOREHI at which GO HIGH is located
	FACLO	Distance after event ISTORELO at which GO LOW is located
	GOLO	Amount of GOLOW range remaining for depenetration
ICLASS	IBOMBER	Bomber class index
	ITANKER	Tanker class index

Table 7. (Part 8 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
IDP	IDP(2)	Depenetration corridor index number as reassigned when last target is an ASM target
IFSCOM	ISTPAR	Number of SETTING/IF clause pairs for STRIKE tape
	JSIF(100)	Pointers to IF clauses for STRIKE tape
	JSSET(100)	Pointers to SETTING clause for STRIKE tape
	IABPAR	Number of SETTING/IF clause pairs for ABTAPE
	JABIF(100)	Pointers to IF clauses for ABTAPE
	JABSET(100)	Pointers to SETTING clauses for ABTAPE
IFUNC	JFUNC(20)	Function codes input (alphabetic)
	INDFUNC(20)	Numeric function codes
IGO	IGO800	Set to 1 for degenerate target area
INDATA	INDATA	Side (1=Blue, 2=Red)
	INBASE	Launch base index
	INDV	Vehicle index
	ILAST	Number of MIRVs per missile
	ITYPE	Weapon type index
	ICLZSS	Weapon class index
	IRZG	Weapon group region
	IZLERT	Weapon group alert status

Table 7. (Part 9 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
IOUT	LPAYLOAD	Index of current payload
	LREF	Index of current refuel area
	LDPEN	Index of current depenetration point
	KOKO	Index of current ASM type
	JFCTNO	Function code of current vehicle
IRF	IRF	Assigned refuel area index
	NRF	Number of refuel areas
IRFTK	IRFTK	Refuel area index
KEYLENG	LOS	Length of /OUTSRT/
	LIN	Length of /INDATA/
	LDN	Length of DINDATA/
	LINC	Length of /INDATA/ except for last array
KEYS	KEYDPEN	Key for packing depenetration point
	KEYBMX	Key for packing recovery point
LASM	U1	Latitude of beginning point of bomber path
	V1	Longitude of beginning point of bomber path
	U2	Latitude of end point of bomber path
	V2	Longitude of end point of bomber path
	UAT	Latitude of ASM target
	VAT	Longitude of ASM target

Table 7. (Part 10 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
LASM (cont.)	RASM	Range of ASM
	RLAT	Latitude of ASM aim point
	RLONG	Longitude of ASM aim point
LASREF	LASREF	Reference Code (IDS) of last non-tanker sortie
LAUNSNAP	INRANGE	Set to zero if ASM target is in range of flight path; otherwise to one
	FRACPATH	Fraction of total path at which ASM is launched
MH	MHMINA(10)	Line in common /DINDATA/ where target area begins
	MHMAXA(10)	Line in common /DINDATA/ where target area ends
	MHMN	Lower plot marker for sortie
	MHMX	Upper plot marker for sortie
MH2	MHMIN(2)	Lower plot markers for sortie
	MHMAX(2)	Upper plot markers for sortie
MISCT	MISCT	Missile booster count
	MTARGCT	Missile target count
MODE	MODE	1 for high altitude, 4 for low altitude
MRVFLG	MRVFLG	Set to 1 if plan contains MIRVs
NOFSYS	NOFSYS(100)	Offensive system type index
OUTSRA	DISTE(10)	Effective distance between target
	HOB(10)	Height of burst information

Table 7. (Part 11 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
OUTSRA (cont.)	DLTA(10)	Change in time information
	INDR(10)	Change indicator for targets
	IOHOB	Height of burst flag
	ICTIME	Change time flag
OUTSRT	ISORTN	Sortie Number
	IOUTSRT	Sortie Number
	MYGROUP	Weapon group index
	MYCORR	Penetration corridor index
	INDVEH	Vehicle index
	IREF	Refuel index
	IDPEN	Depenetration corridor index
	IPAYLOAD	Payload table index
	LNCHBASE	Launch base index number
	ITYP	Weapon group type index
	BLAT	Launch base latitude
	BLONG	Launch base longitude
	NHAP	Number of sortie events
	IBTYPE(14)	Event type
	OBLAT(14)	Latitude of event
	OBLONG(14)	Longitude of event
	DLAT(14)	Target offset - in degrees latitude
	DLONG(14)	Target offset - in degrees longitude

Table 7. (Part 12 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
OUTSRT (cont.)	IBJEC(14)	Event place code
	IBDES(14)	Target DESIG
	IBTSK(14)	Task code of target
	IBCTY(14)	Country code of target
	IBFLG(14)	Target FLAG
	ATTROUT(14)	Local attrition
	SURVOUT(14)	Cumulative survival probability
	LXMYHOB(1)	Height of burst indicator
	GOLOW1	Low altitude range available for use in corridor
	GOLOW2	Low altitude range available for use before first target
	GOLOW3	Low altitude range available for use after first target
	SPDLQ	Speed at low altitude
	SPDHI	Speed at high altitude
	RANGX	Range of vehicle
	RANGREF	Refueled range of vehicle
	DELAY	Delay of vehicle launch
	IRG	Vehicle launch region
	ILRT	Vehicle alert status
	IDBOMBER	Vehicle identification
	AVAILOW	Available low altitude range

Table 7. (Part 13 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
OUTSRT (cont.)	RNGDEC	Range decrement at low altitude
	DRECOVER	Distance to recovery
	DISTLEGO	Distance to origin
PAYSTF	NOBOMB1(40)	Number of bombs of type 1 (number of RVs for missiles)
	IWHD1(40)	Warhead index of bomb type 1
	NOBOMB2(40)	Number of bombs of type 2
	IWHD2(40)	Warhead index of bomb type 2
	NASM(40)	Number of ASMs
	IASM(40)	Warhead index of ASM
	NPCM(40)	Number of counter measures
	NPDCY(40)	Number of decoys
	NAPDCY(40)	Number of area decoys
	IMIRV(40)	MIRV system identifier
	PYALT(40)	Weapon release altitude (bombers)
PLTYP	IPLANTYP	Weapon plane type index
	PLANAME	Weapon type name (6 characters)
	SHORNAME	Weapon type name (3 characters)
POLITE	S1	Latitude of beginning interpolation point
	T1	Longitude of beginning interpolation point
	S2	Latitude of interpolation end point
	T2	Longitude of interpolation end point

Table 7. (Part 14 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
POLITE (cont.)	FACTOR	Interpolation factor or fraction
	SR	Latitude of interpolated point
	TR	Longitude of interpolated point
PPINFO	NDUMCORR	Tactical aircraft corridor index
	GOLOX1	Saved low altitude range available for use in corridor
	GOLOX2	Saved low altitude range available for use before first target
	GOLOX3	Saved low altitude range available for use after first target
	INDEX	Bomber plan index equals group number plus 100 times corridor index plus 10000 times sortie number
	NSORTIES	Total number of bomber plans processed
	INDXX	Group weapon type index
	GOGO	Saved low altitude range available for use in corridor
	MHIST	Maximum number of entries into history table
	DUST	Distance bomber traveled during first history event
PPXX	LXIDPCHK	Logical area indicating if depenetration corridor is used.
	ILAUNDEX(90)	Number of decoys launched
	TIMELAUN(90)	Time of decoy launch
	DISTORE(90,6)	Distance traveled by decoy
	HDTX(90)	Temporary line array

Table 7. (Part 15 of 20)

<u>Block</u>	<u>Array and Variable</u>	<u>Description</u>
PPXX (cont.)	KPLX(90)	Temporary place array
	JTPX(90)	Temporary event number array
	HLAX(90)	Temporary latitude array
	HLOX(90)	Temporary longitude array
	TZTX(90)	Temporary offset latitude array
	TZNX(90)	Temporary offset longitude array
	IWHX(90)	Temporary warhead index array
	PAY(90)	Temporary probability of arrival array
	CMTX(90)	Temporary cumulative time array
PRNCON	INDEXPR(15)	Print request number
	JAGROUP(15)	First group for request
	JACORR(15)	First corridor for request
	JSSORT(15)	First sortie for request
	LAGROUP(15)	Last group for request
	LACORR(15)	Last corridor for request
	LASORT(15)	Last sortie for request
	KFREQ(15)	Frequency for request
PSW	NREQ	Number of requests
	STPRIN	True if STRIKE tape print requested
	ABPRIN	True if ABTAPE print requested
	ABUNIT	Report code for ABTAPE print

Table 7. (Part 16 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
RECBAS	RCBLAT(50,4)	Recovery base latitude*
	RCBLON(50,4)	Recovery base longitude*
	INDBAS(50,4)	Recovery base name*
	INDCAP(50,4)	Recovery base capacity*
	DISTR(50,4)	Distance to recovery*
	TOF(50,4)	Time of flight to recovery*
	RECLAT(50)	Tanker recovery latitude
	RECLONG(50)	Tanker recovery longitude
RECOVERY	NAMECAP(200,3)	J=1, Recovery base name J=2, Recovery base capacity J=3, Number of aircraft that launched at recovery base
	NUSED(200)	Working array that defines the number of aircraft arriving at each base for a given group
	IREC	Logical unit containing recovery base data
	NREC	Number of unique recovery bases
	NBOMGP	Number of bomber groups
RL	NMSGRP	Number of missile groups
	RL	Decoy low-altitude range
	RH	Decoy high-altitude range
SNAPON	NAP(15)	Set to three for active print I; set to one for inactive print I

* Indexed for each of four bases assigned to each depenetration point.

Table 7. (Part 17 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
TANKA	IINDEXTK(60)	Tanker base index
	TKRLAT(60)	Latitude of tanker base
	TKRLONG(60)	Longitude of tanker base
	IIREFTK(60)	Refuel area for tankers where $N > 0$ implies must refuel at area n
	NTKPSQN(60)	Number of tankers in squadron at base
	NALRTNK(60)	Number of alert tankers at base
	TANKSPP(60)	Speed of tankers at base
	TKDLYALT(60)	Delay for alert tankers at base
	TKDLYLN(60)	Delay for non-alert tankers at base
	TKTTOS(60)	Total time on station
	IITYPTK(60)	Tanker type index
	TRANGX(60)	Tanker range
TANKB	COST(60,50)	Distance between tanker base I and refuel area J
	SOURCE(60)	Number of tankers at base I to be automatically assigned
	ISOL(110)	The Ith nonzero element in the final VAM solution
	RBASLOC(110)	Tanker base corresponding to the Ith solution element
	CBASLOC(110)	Refuel area corresponding to the Ith solution element
	NSOL	Number of nonzero elements in VAM solution

Table 7. (Part 18 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
TANKB (cont.)	RMAX	Number of rows (tanker bases) in VAM problem
	LXIRCHK(2)	Logical array true if base not automatically allocated
	IRCDIF	Number of bases for which LXIRCHK is true
	DISTREF(50)	Distances from current tanker base to refuel area I
TANKER	INDEXTK	Tanker index
	TKLAT	Tanker latitude
	TKLONG	Tanker longitude
	IREFLK	Tanker refuel area index
	NPSQNTK	Number of tankers per squadron
	NALRTK	Number of alert tankers
	SPEEDTK	Tanker speed
	DLYALTK	Delay for alert tankers
	DLYNLTK	Delay for nonalert tankers
	TTQS	Total time on station
	ITYPETK	Tanker type index
	TANKRNGE	Tanker range
TEMPO	DT(50)	Distance on time temporary storage
	JT(50)	Event type temporary storage
	TLT(50)	Latitude temporary storage
	TLN(50)	Longitude temporary storage
	LPI.(50)	Place index temporary storage

Table 7. (Part 19 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
TIMELINE	LXITIM(2)	Packed logical array: True if type of CORMSL is "line"
	CORMSLX(40)	Percent flight complete or time on line
	ZLAT(50,2)	Latitude of timing end points
	ZLONG(50,2)	Longitude of timing end points
	XC(50)	X-coordinate of cross product vector of lining line
	YC(50)	Y-coordinate of cross product of timing line
	ZC(50)	Z-coordinate of cross product of timing line
	DL(50)	Length of timing line
	NLINES	Number of timing lines
TYPSTF		Weapon type data
	TRANGE(100)	Weapon range
	TCEP(100)	Weapon CEP
	TTOFMIN(100)	Weapon minimum time of flight
	TCMISS(100)	Weapon missile constant
	TRNGMN(100)	Weapon minimum range
	TREL(100)	Weapon reliability
	TNAME(100)	Weapon name
	TFUNCT(100)	Weapon function code
	TLINT(100)	Weapon launch interval
	ISMLUN(100)	Number of weapons which may be launched simultaneously

Table 7. (Part 20 of 20)

<u>Block</u>	<u>Array or Variable</u>	<u>Description</u>
VICINITY	VHB	Bomber cannot go high within VHB miles before target
	VHA	Bomber cannot go high within VHA miles after target
	VLB	Bomber cannot go low within VLB miles before target
	VLA	Bomber cannot go low within VLA miles after target
	GOMIN	Bomber cannot fly low for less than GOMIN minutes
WHDSTF	WHDYLD(50)	Warhead yield
	WHDFRC(50)	Warhead fission/fusion fraction
	WHDRNG(50)	Warhead range*
	WHDREL(50)	Warhead reliability*
	WHDCEP(50)	Warhead CEP*
	WHDSPD(50)	Warhead speed*

* ASM warhead only

4.7 Subroutine ENTMOD

PURPOSE: Driver subroutine for PLANOUT Module

ENTRY POINTS: ENTMOD (first subroutine called when overlay PLANOUT is executed)

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, C15, C30, CALLSW, LASREF, PRNCON, ZEES

SUBROUTINES CALLED: ABORT, ALTPLAN, HDFND, INITANK, INSGET, INTRFACE, PLANTANK, PLNTPLAN, RETRV

CALLED BY: MODGET

Method:

First the switches in the /CALLSW/ block are set to false. These switches prevent the routines GEOGET, LNCHDATA and WEPDATA from being called more than once. Next the NUMTBL record is retrieved. The reference code of the last nontanker sortie (LASREF) is obtained from the reference code of the last bomber sortie (LMBBREF) or from the reference code of the last missile sortie (LMSLREF) if LMBBREF is zero.

Now all adverbs are read and switches and pointers in block /ADVRB/ are set accordingly. The ONPRINTS clause is now processed to set values in the /PRNCON/ block.

Finally, the appropriate overlays are read in and executed according to switches set by adverbs. If RECALC was input, INITANK, PLNTPLAN and PLANTANK are called. If ACARD, CCARD or ICARD clauses were included, ALTPLAN is called. And, if STRIKE or ABTAPE were input, INTRFACE is called.

Subroutine ENTMOD is illustrated in figure 62.

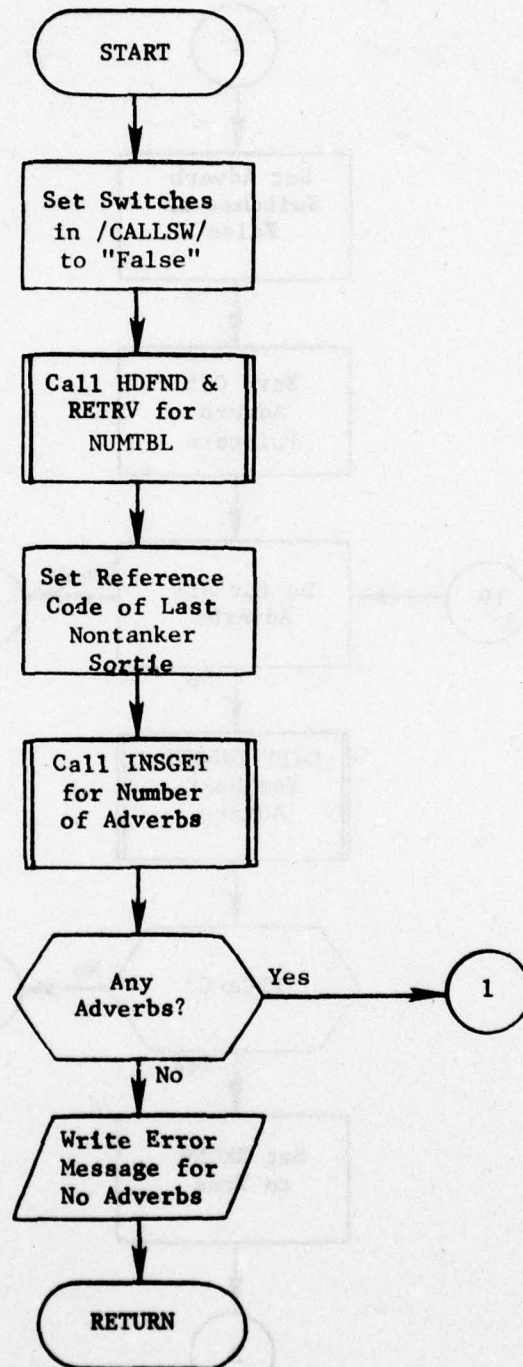


Figure 62. Subroutine ENTMOD (Part 1 of 13)

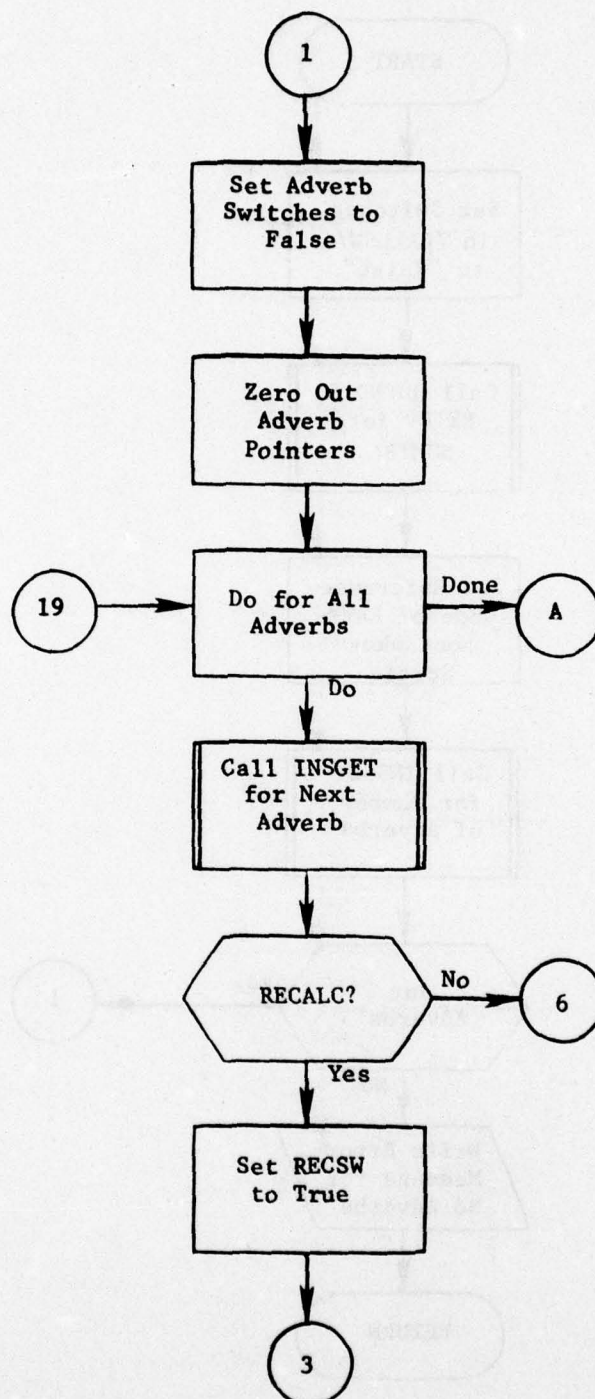


Figure 62. (Part 2 of 13)

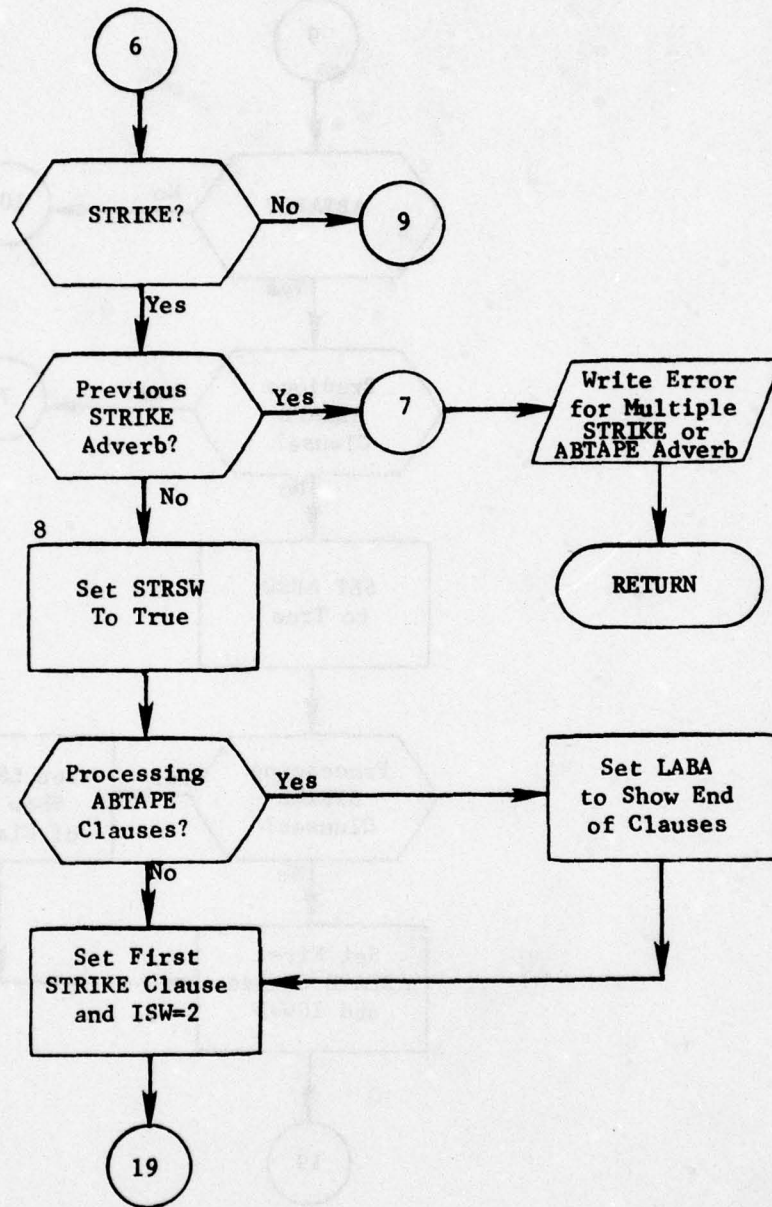


Figure 62. (Part 3 of 13)

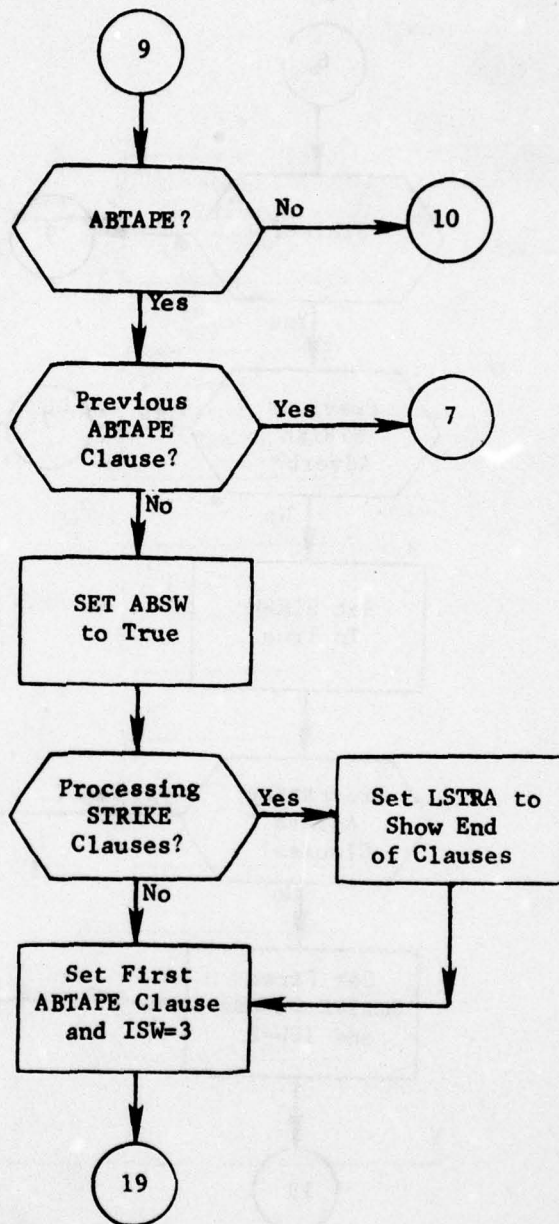


Figure 62. (Part 4 of 13)

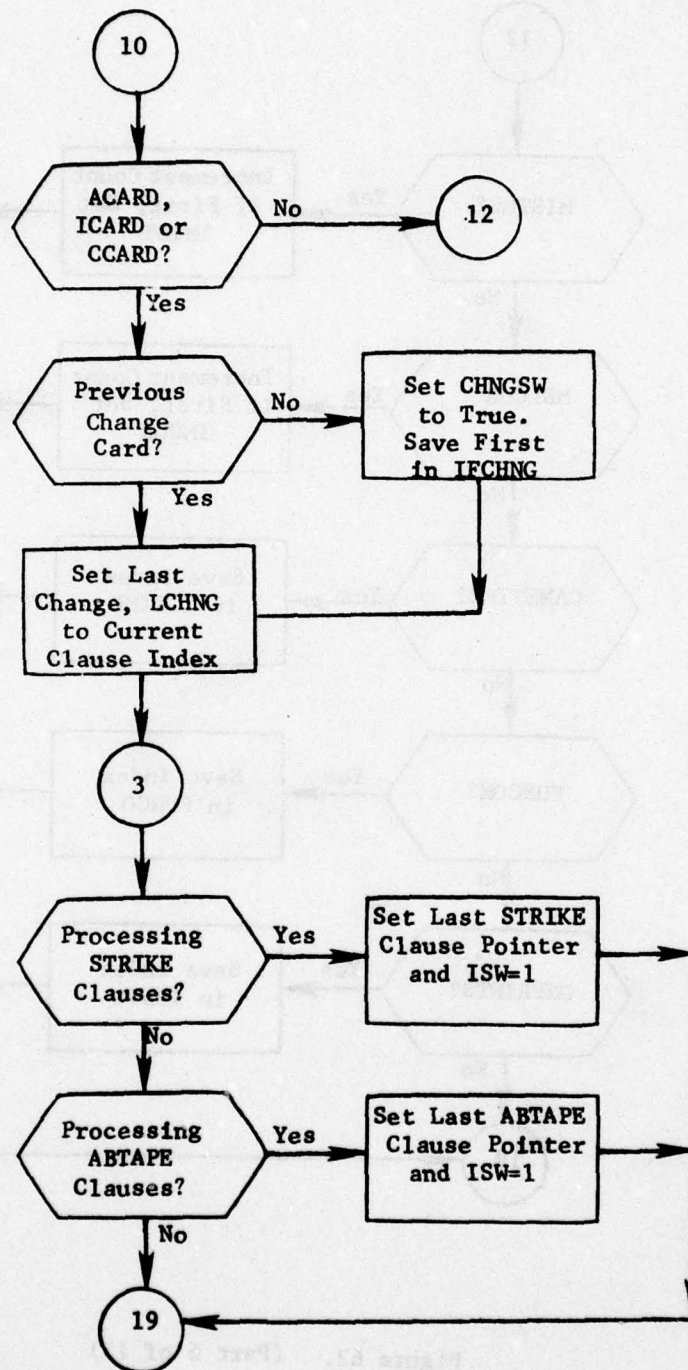


Figure 62. (Part 5 of 13)

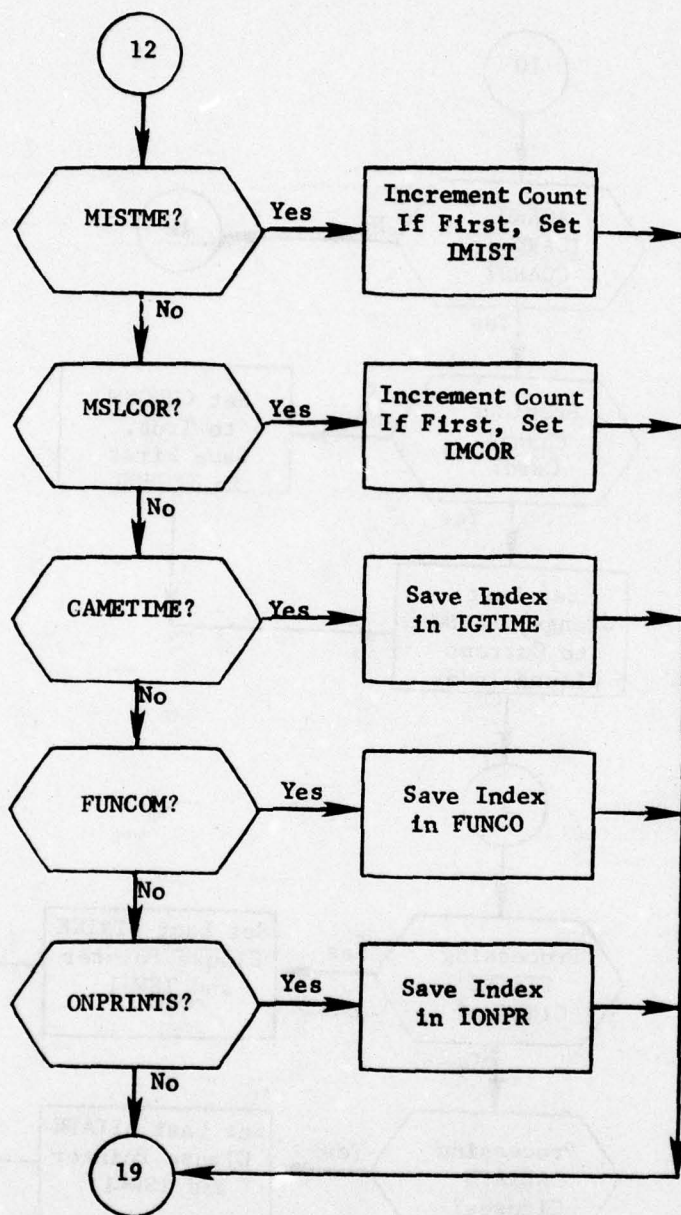


Figure 62. (Part 6 of 13)

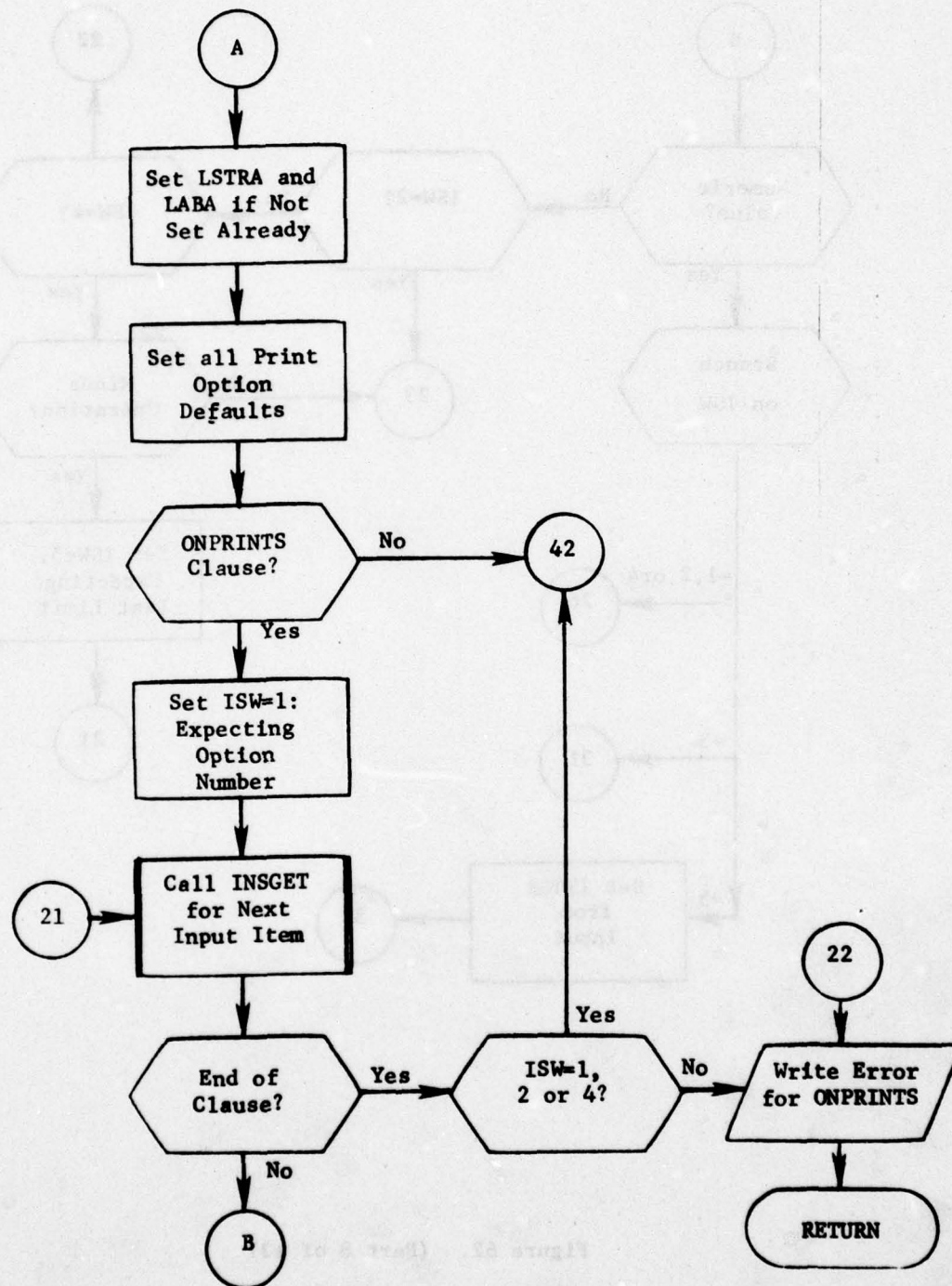


Figure 62. (Part 7 of 13)

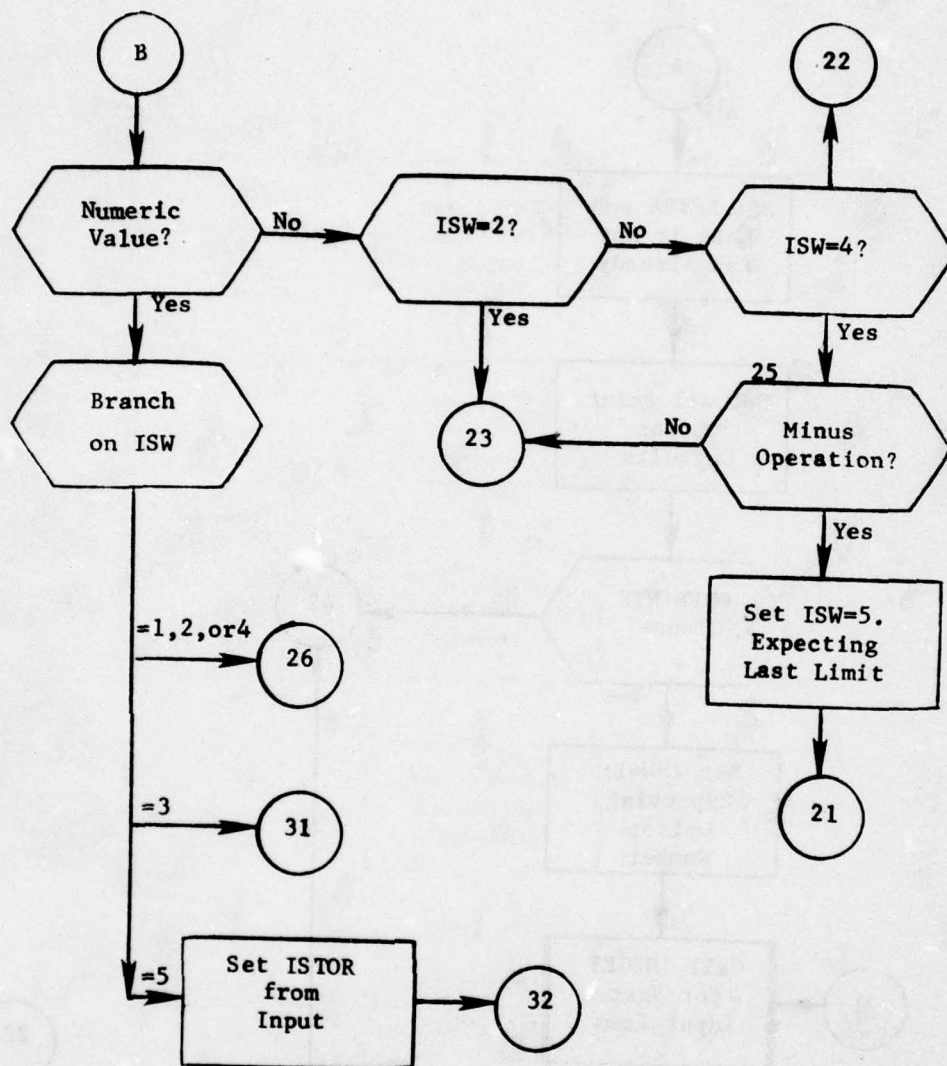


Figure 62. (Part 8 of 13)

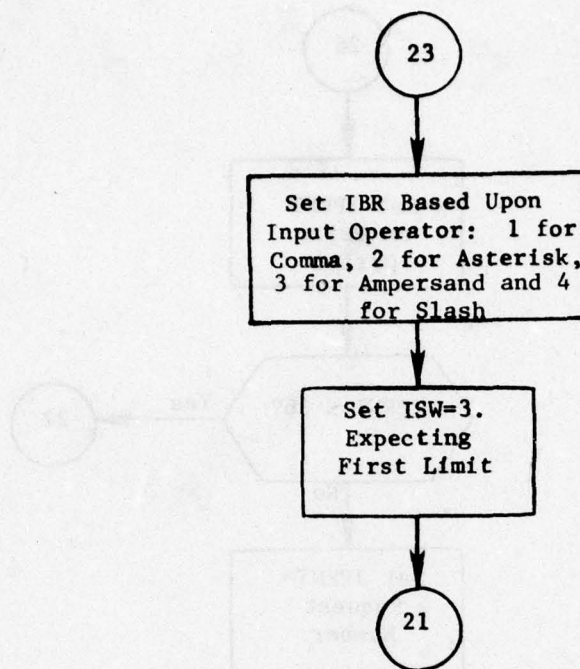


Figure 62. (Part 9 of 13)

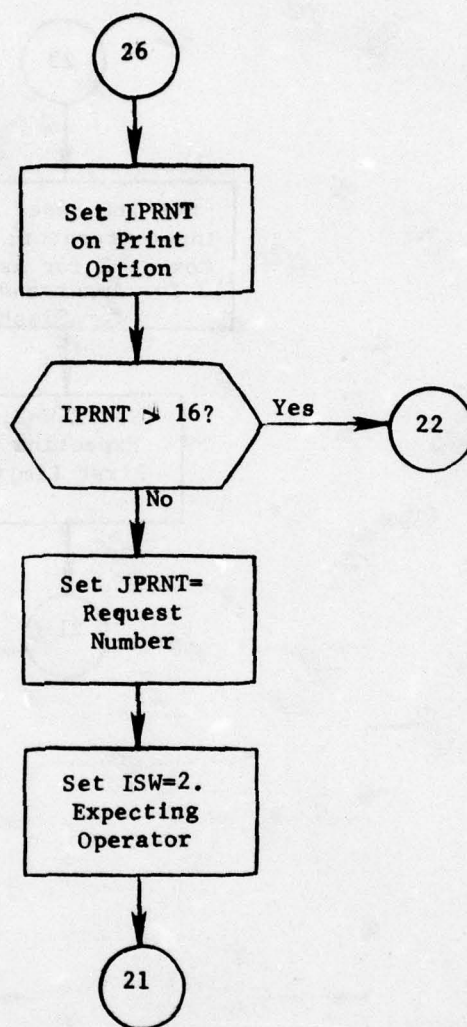


Figure 62. (Part 10 of 13)

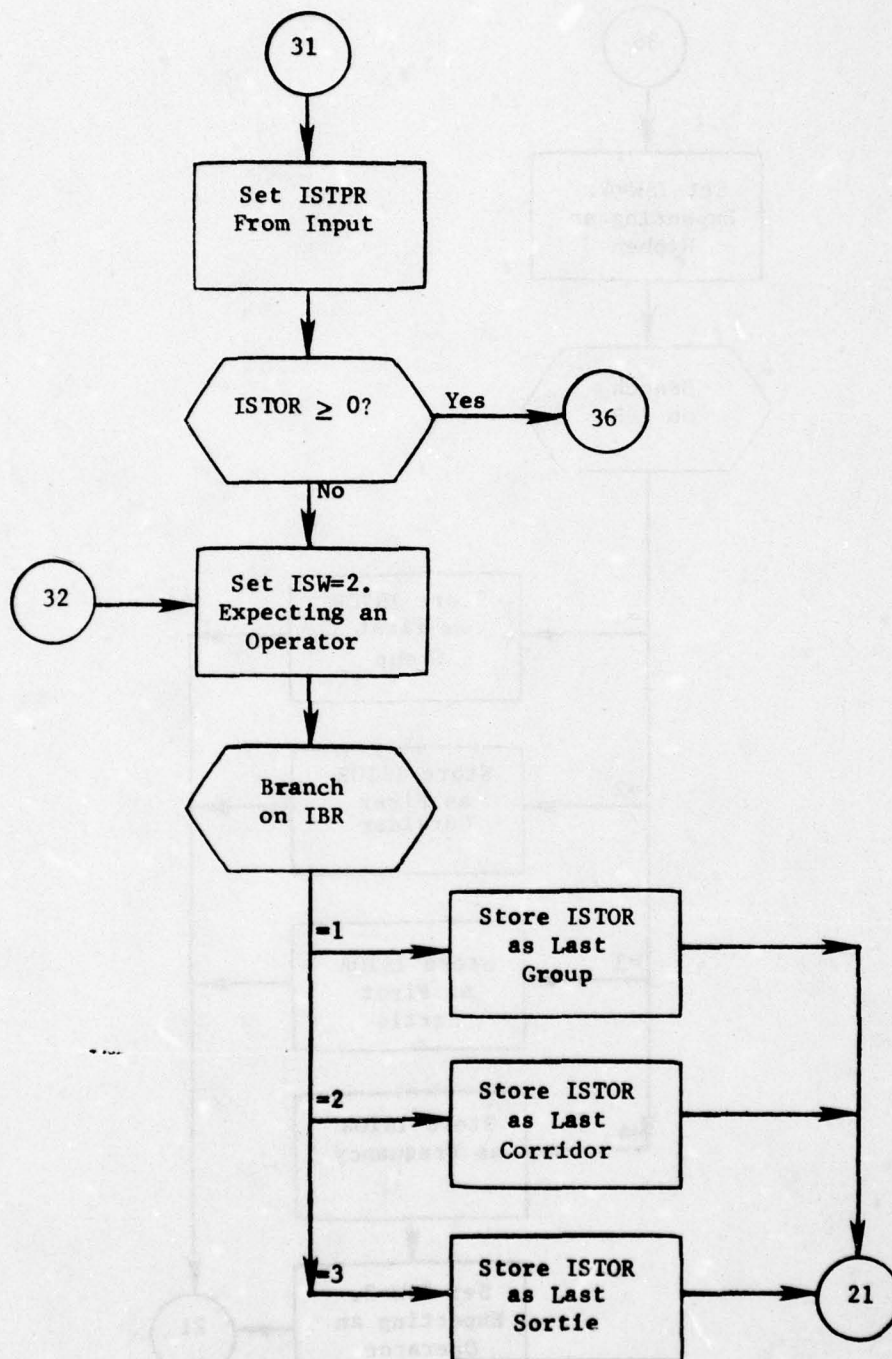


Figure 62. (Part 11 of 13)

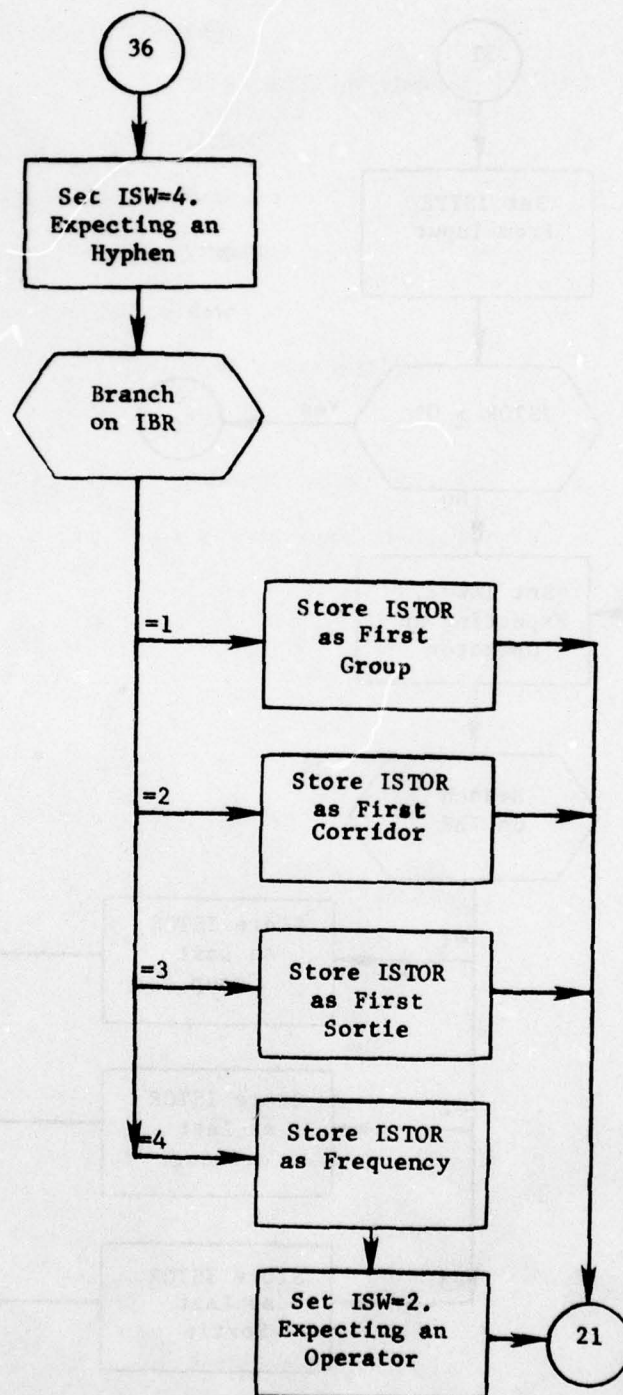


Figure 62. (Part 12 of 13)

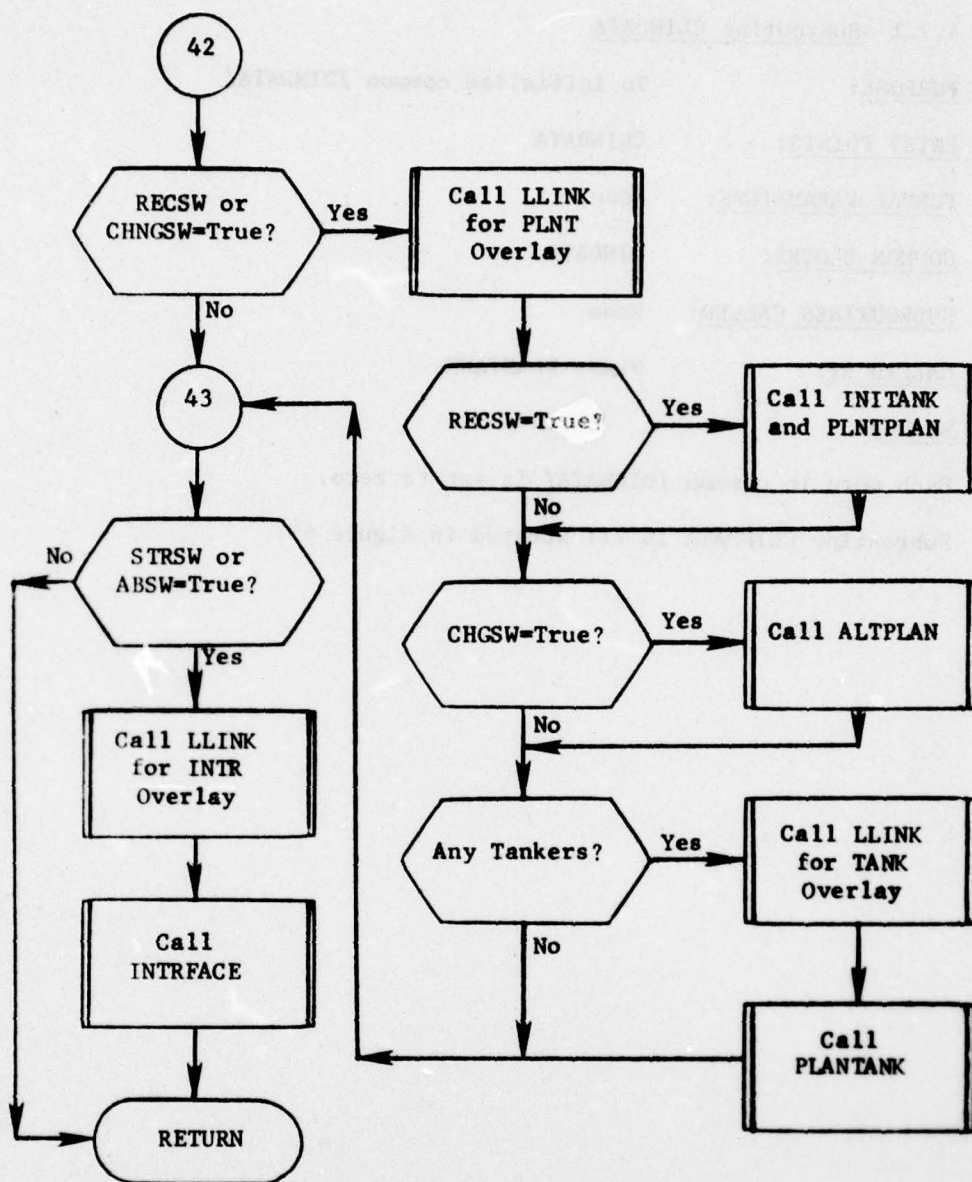


Figure 62. (Part 13 of 13)

4.7.1 Subroutine CLINDATA

PURPOSE: To initialize common /DINDATA/

ENTRY POINTS: CLINDATA

FORMAL PARAMETERS: None

COMMON BLOCKS: DINDATA

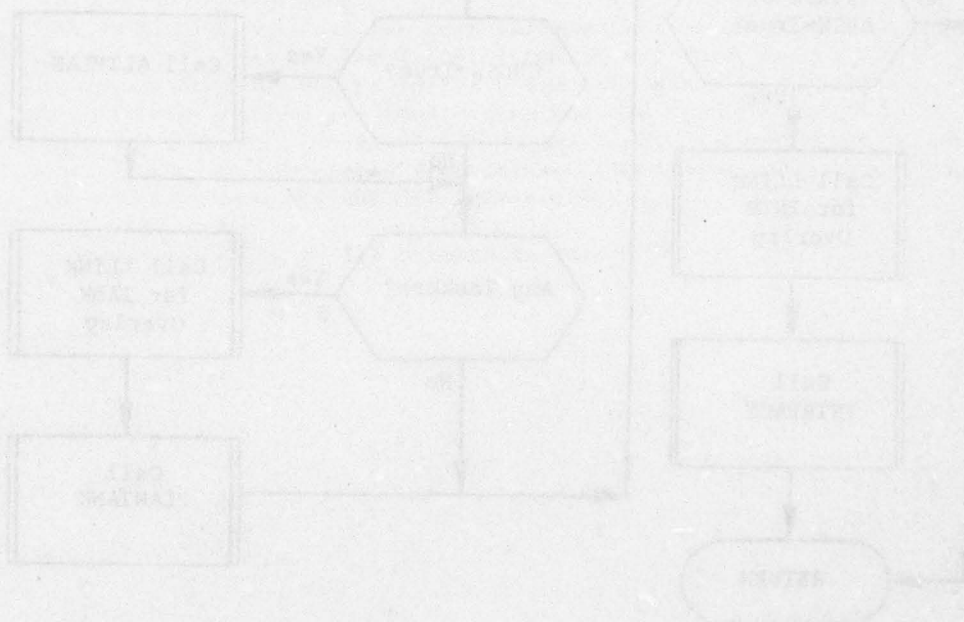
SUBROUTINES CALLED: None

CALLED BY: PLAN, PLANTANK

Method:

Each word in common /DINDATA/ is set to zero.

Subroutine CLINDATA is illustrated in figure 63.



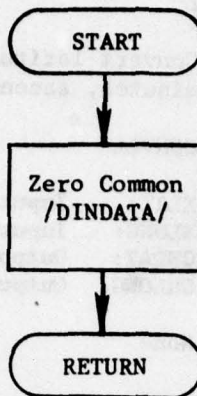


Figure 63. Subroutine CLINDATA

4.7.2 Subroutine CONVLL

PURPOSE: Convert latitude and longitude to degrees, minutes, seconds (DMS) format

ENTRY POINTS: CONVLL

FORMAL PARAMETERS: XLAT: Input latitude
XLONG: Input longitude
CHLAT: Output latitude (character *7)
CHLONG: Output longitude (character *8)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: ABOUT, SNAPOUT, STOUT

Method:

The process is similar for both latitude and longitude. The latitude is converted first. The letter (CHM) is set to N and if the latitude is negative it is set to positive and CHM is set to S. The degrees, minutes and seconds are then broken out and ENCODED into CHLAT. Longitude is now processed, CHM is set to W. If longitude is greater than 180 it is subtracted from 360 and CHM is set to E. Longitude is then broken down and ENCODED into CHLONG.

Subroutine CONVLL is illustrated in figure 64.

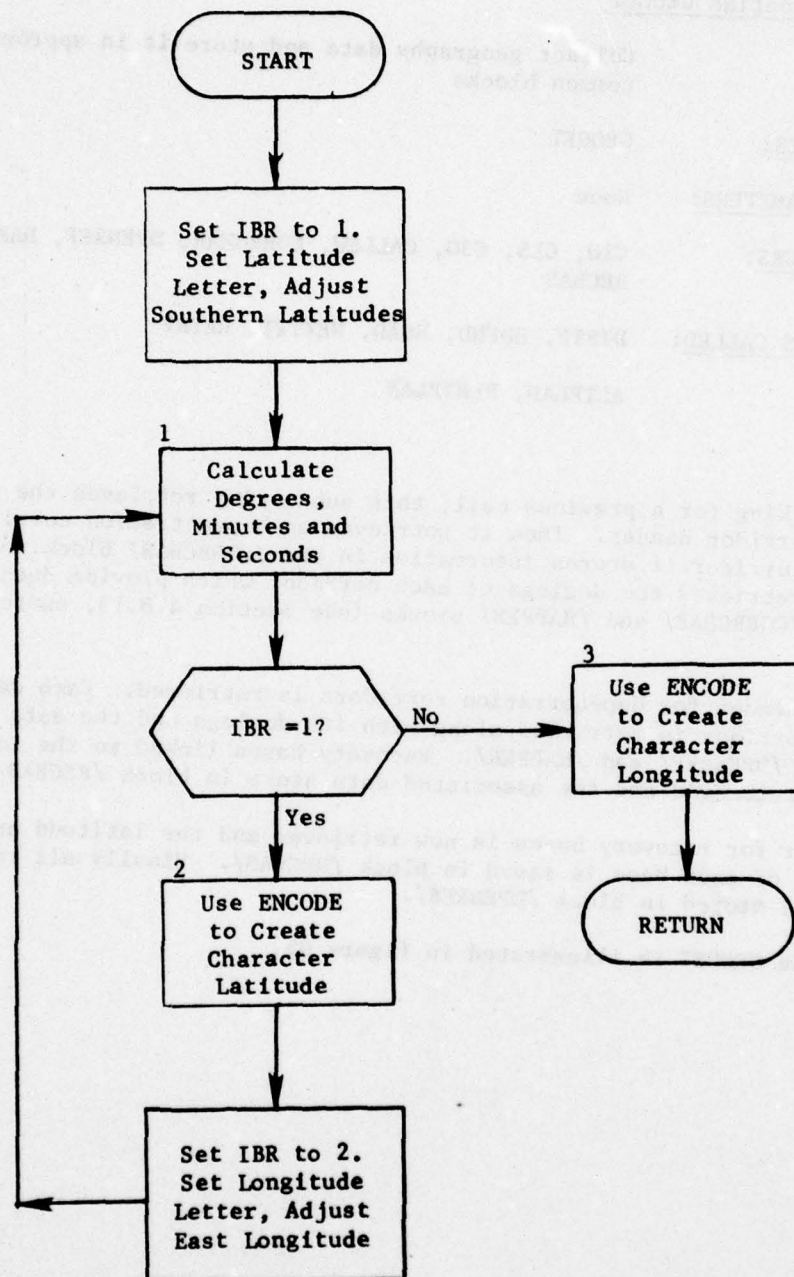


Figure 64. Subroutine CONVLL

4.7.3 Subroutine GEOGET

PURPOSE: Collect geography data and store it in appropriate common blocks

ENTRY POINTS: GEOGET

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, CALLSW, CORRCHAR, DPENREF, HAPPEN, RECBAS

SUBROUTINES CALLED: DISTF, HDFND, HEAD, NEXTTT, RETRV

CALLED BY: ALTPLAN, PLNTPLAN

Method:

After checking for a previous call, this subroutine retrieves the penetration corridor header. Then it retrieves each penetration corridor. For each corridor it stores information in the /CORRCHAR/ block. Further, it retrieves the doglegs of each corridor which provide data for both the /CORRCHAR/ and /HAPPEN/ blocks (see section 4.8.13, subroutine PLAN).

Next the header for depenetration corridors is retrieved. Each depenetration corridor is retrieved along with its doglegs and the data stored in blocks /DPENREF/ and /HAPPEN/. Recovery bases linked to the corridors are also retrieved and the associated data store in block /RECBAS/.

The header for recovery bases is now retrieved and the latitude and longitude of each base is saved in block /RECBAS/. Finally all refuel points are stored in block /DPENREF/.

Subroutine GEOGET is illustrated in figure 65.

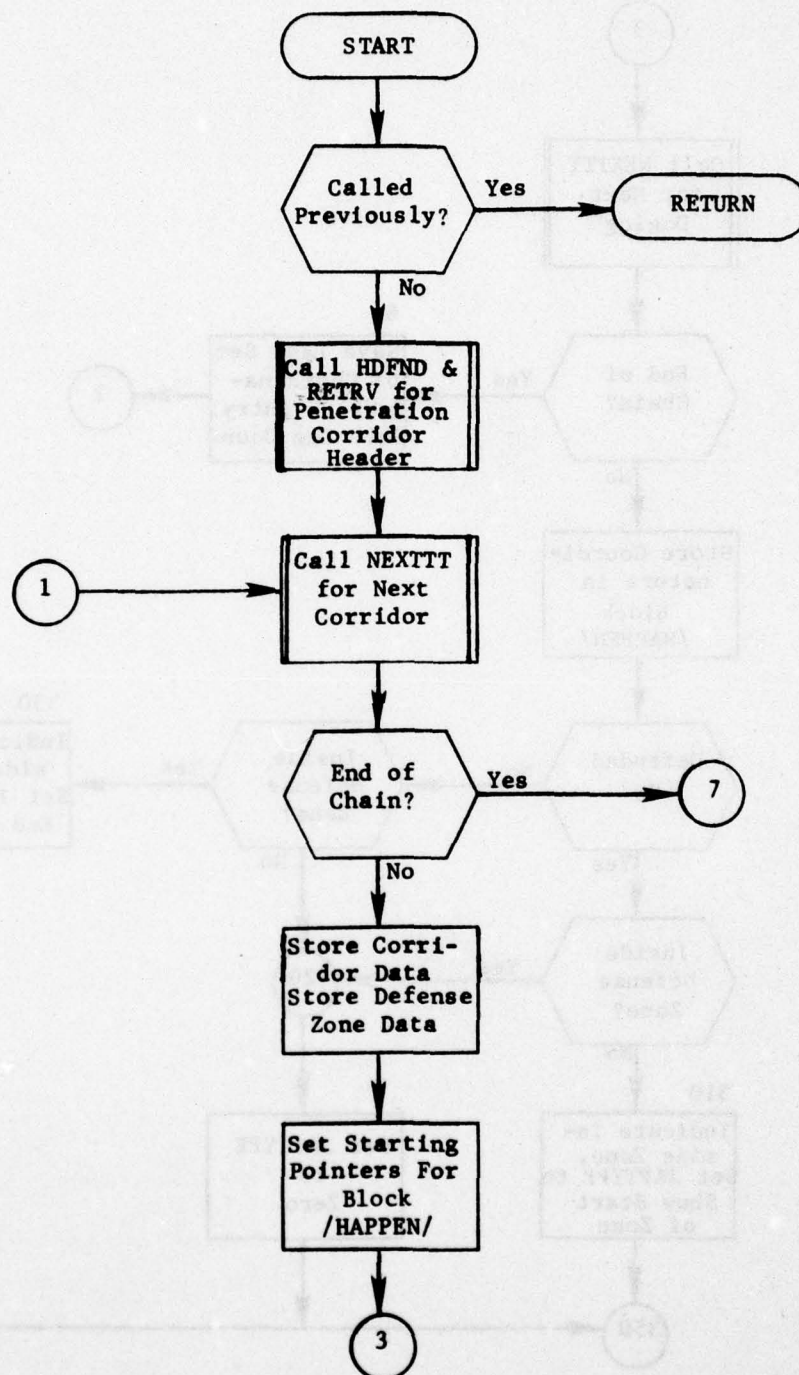


Figure 65. Subroutine GEOGET (Part 1 of 6)

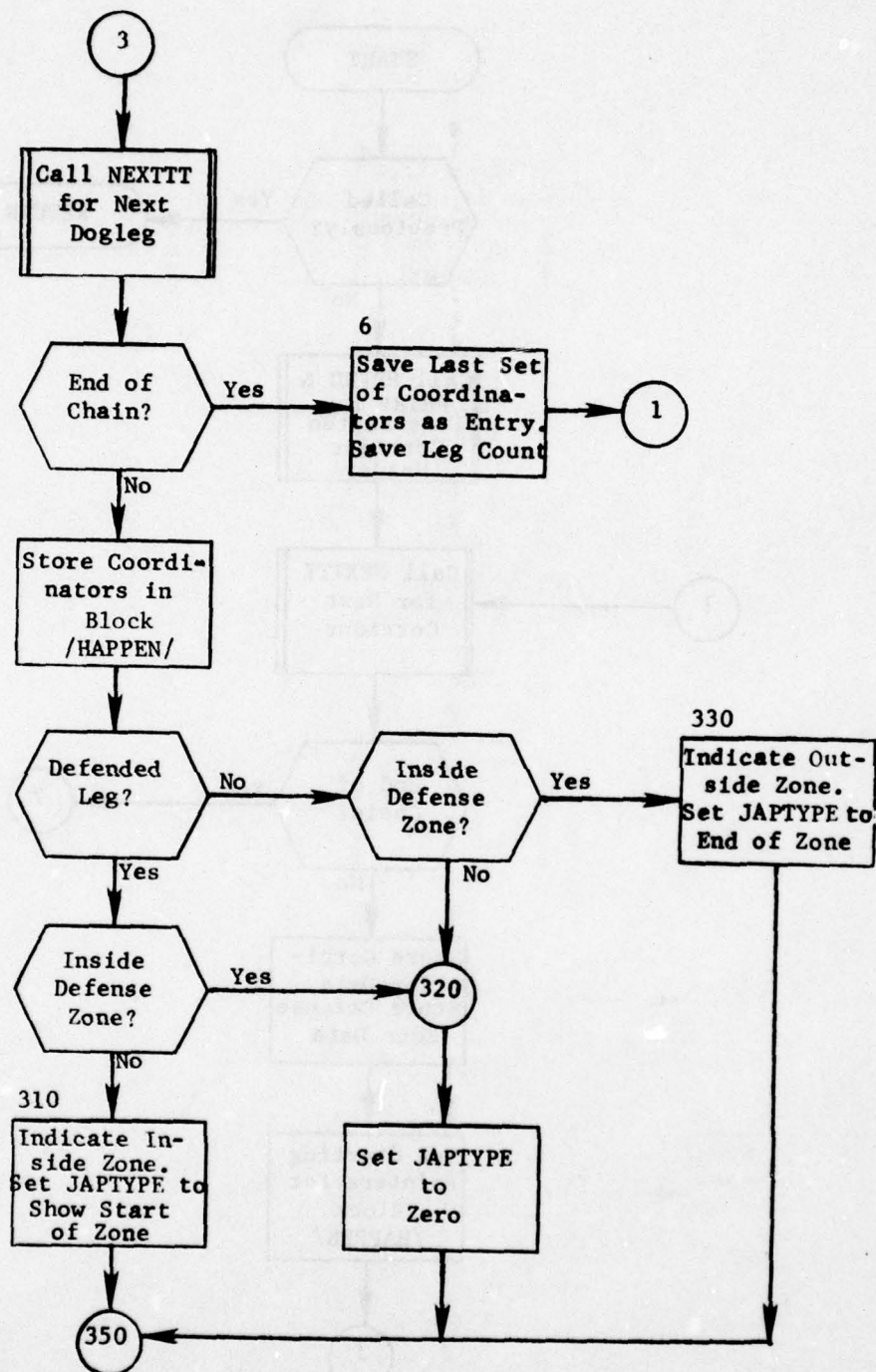


Figure 65. (Part 2 of 6)

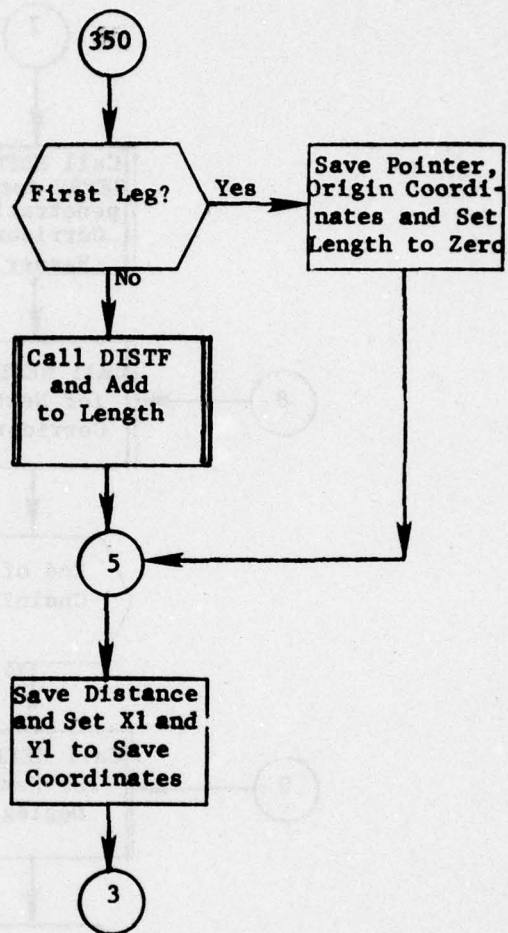


Figure 65. (Part 3 of 6)

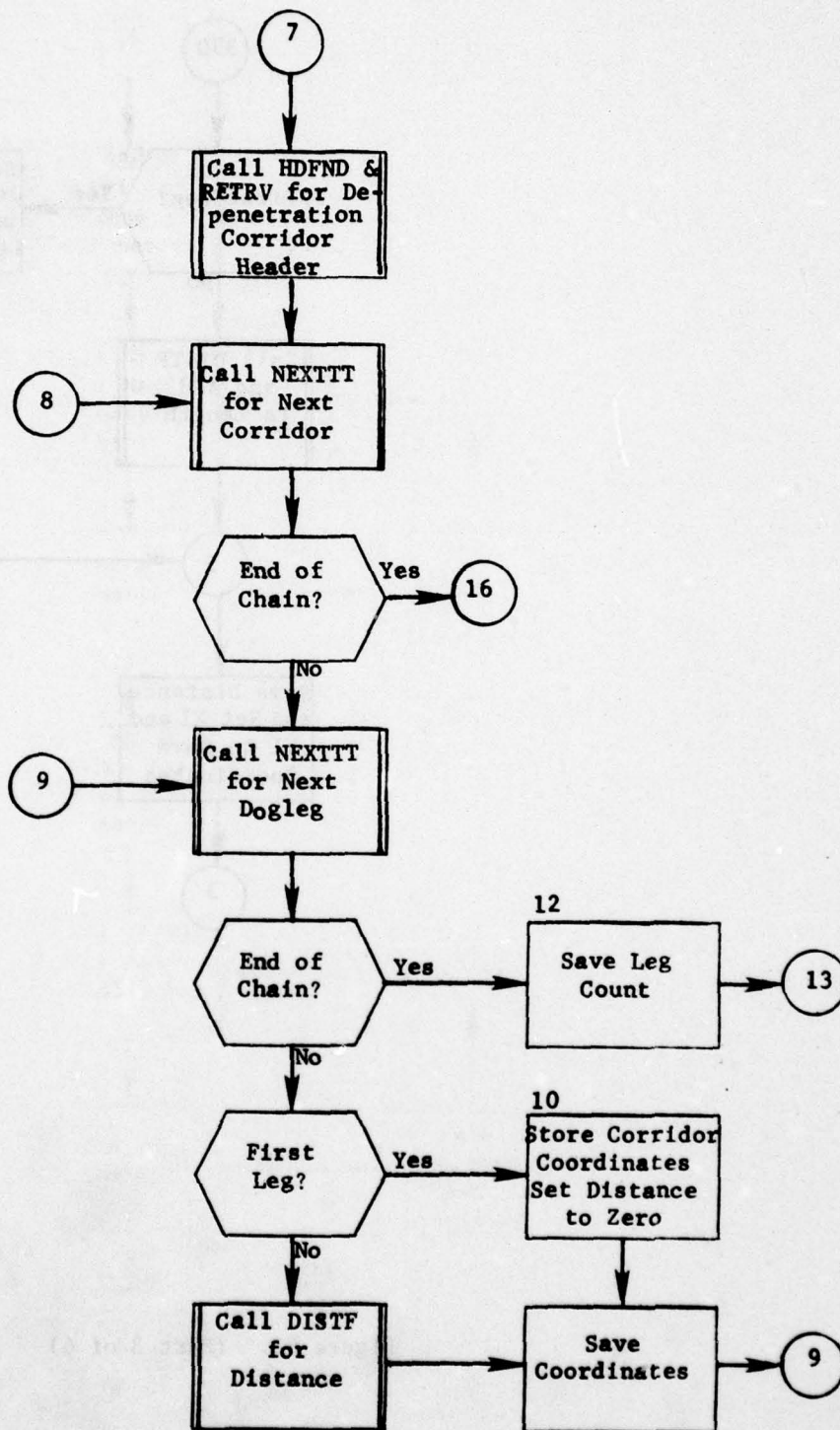


Figure 65. (Part 4 of 6)

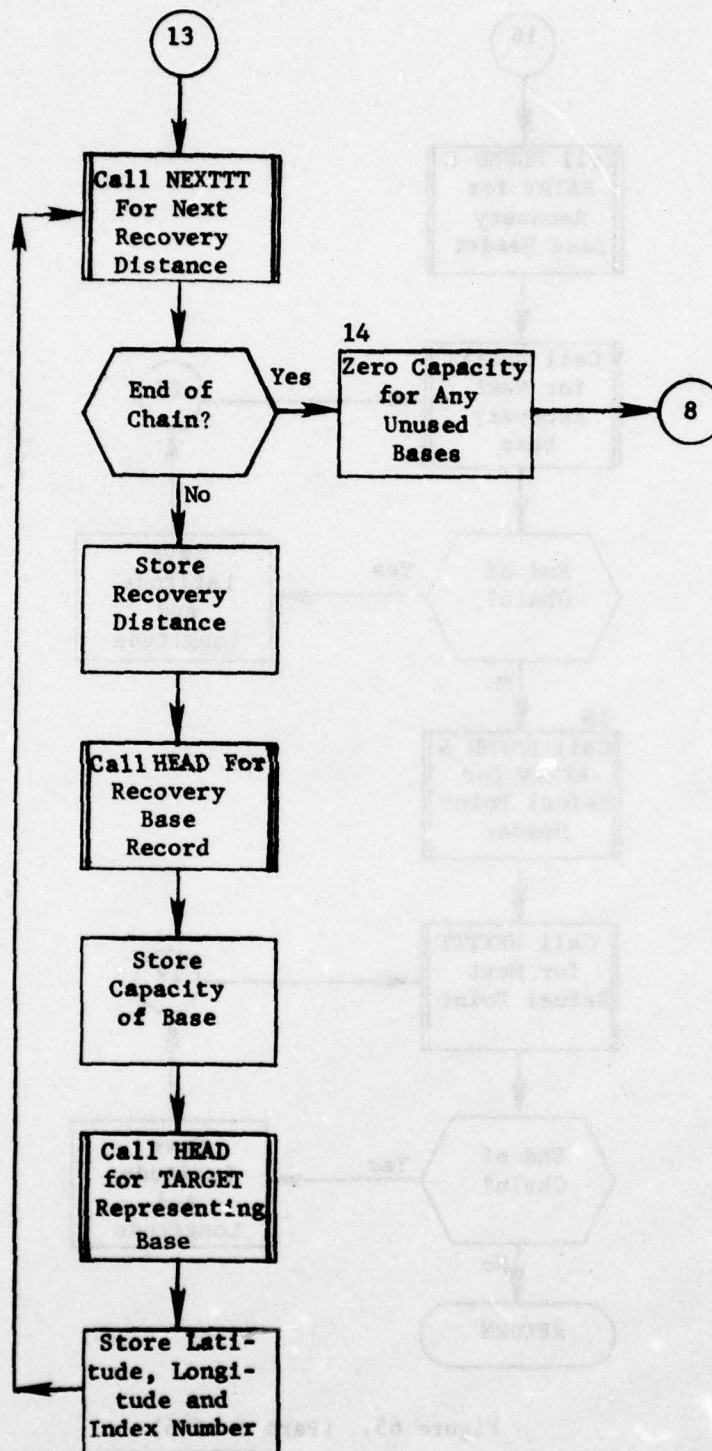


Figure 65. (Part 5 of 6)

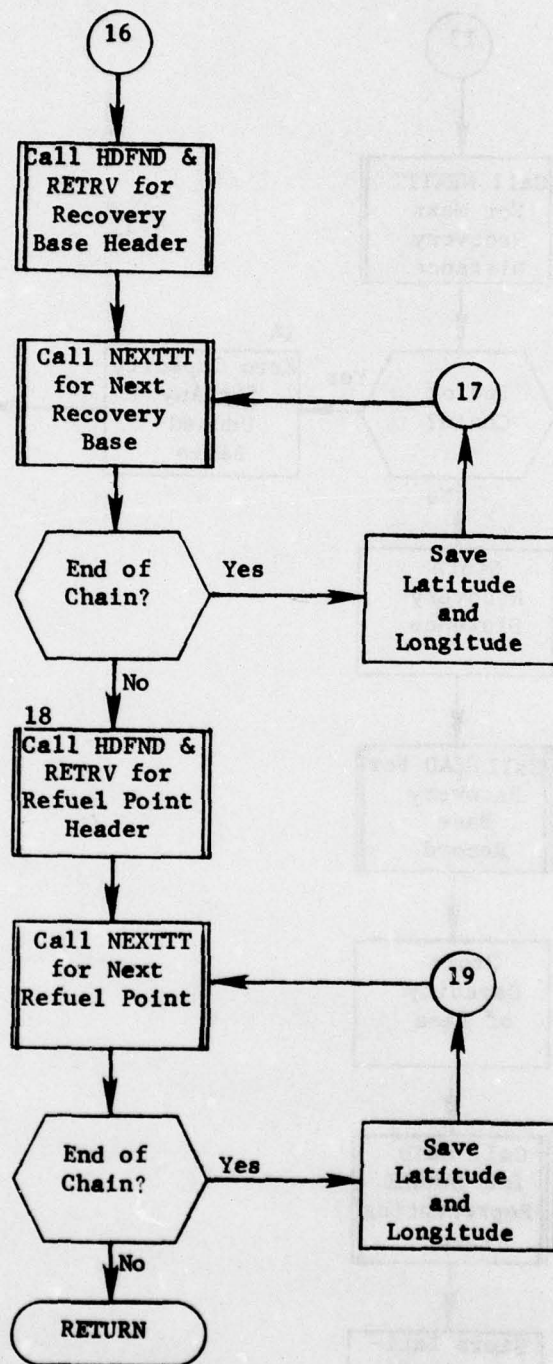


Figure 65. (Part 6 of 6)

4.7.4 Subroutine SNAPCON

PURPOSE: To control the activation of optional prints.
ENTRY POINTS: SNAPCON
FORMAL PARAMETERS: None
COMMON BLOCKS: IFTPRNT, OUTSRT, PRNCON, SNAPON
SUBROUTINES CALLED: None
CALLED BY: ALTPPLAN, PLNTPLAN, PLANTANK

Method:

This subroutine with SNAPIT and SNAPOUT provides the capability for optional printing. SNAPCON uses the ONPRINTS clause information to control the activation of prints during processing. SNAPIT is called wherever an optional print is to be issued; SNAPIT, in turn, calls SNAPOUT to do the actual printing.

For each input sortie, SNAPCON checks the print request list with a particular value of a print control parameter, say group, to determine which prints are to be activated. Let x be the value of the print control parameter; e.g., the group number on the current input record. Suppose that a = the starting group and b = the finishing group as specified on the print request card. Then x is checked to determine whether it is in the interval a to b . Either a or b , or both, may be blank or zero on the control card. Table 8 lists the possible values of a and b , and the value that x should assume if the print is to be active in each of these cases. Let a_m be the minimum value x can have and b_m be its maximum value. Then the following single text of x suffices to determine if x is such that the print is active:

$$a' + a_m L(a') \leq x \leq b' + a_m L(b') + b_m L(a')$$

where $L(x) = 1$ if $x = 0$ and is 0 otherwise.

For each print number (1 to 15) which is to be active for the current plan, SNAPCON sets the corresponding element(s) in the NAP array to 3.

Subroutine SNAPCON is illustrated in figure 66.

Table 8. Possible Values of a and b

<u>a</u>	<u>b</u>	<u>VALUE OF x FOR ACTIVE PRINT</u>
0	0	any value
a'	0	$x = a'$
0	b'	any value
a'	b'	$a' \quad x \quad b'$

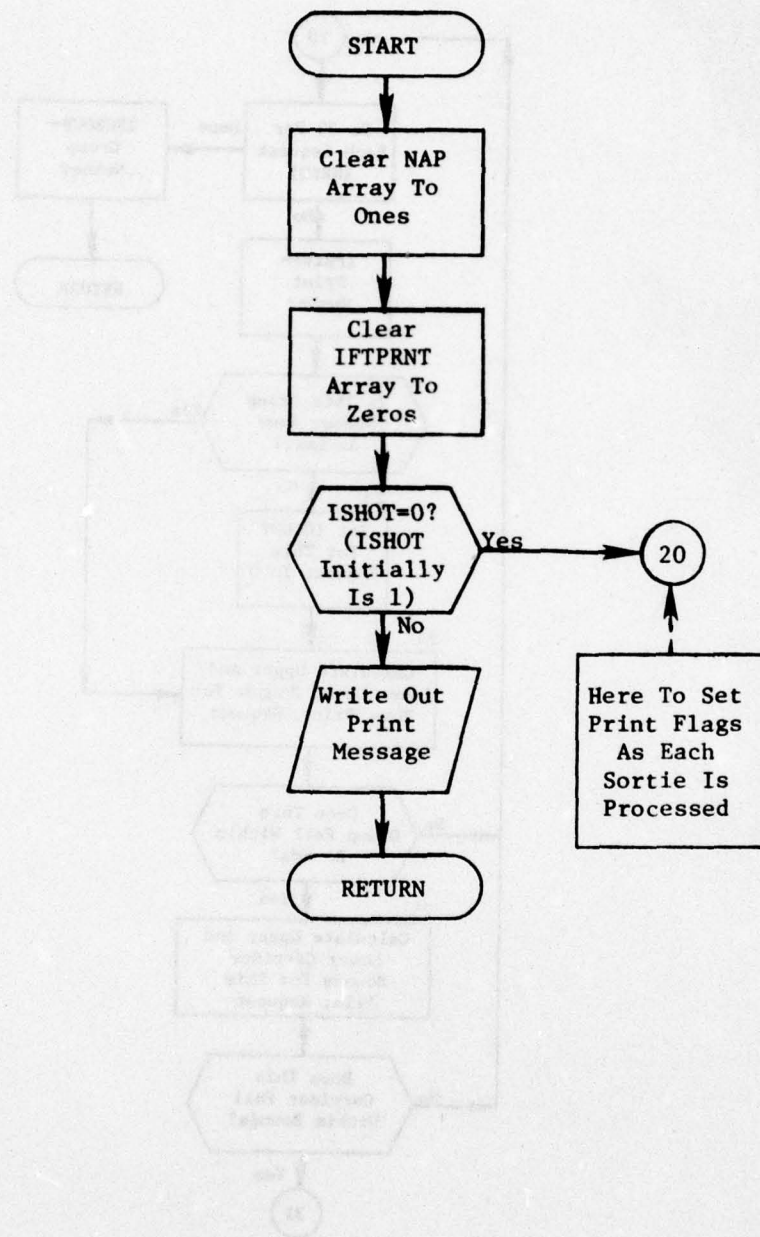


Figure 66. Subroutine SNAPCON (Part 1 of 3)

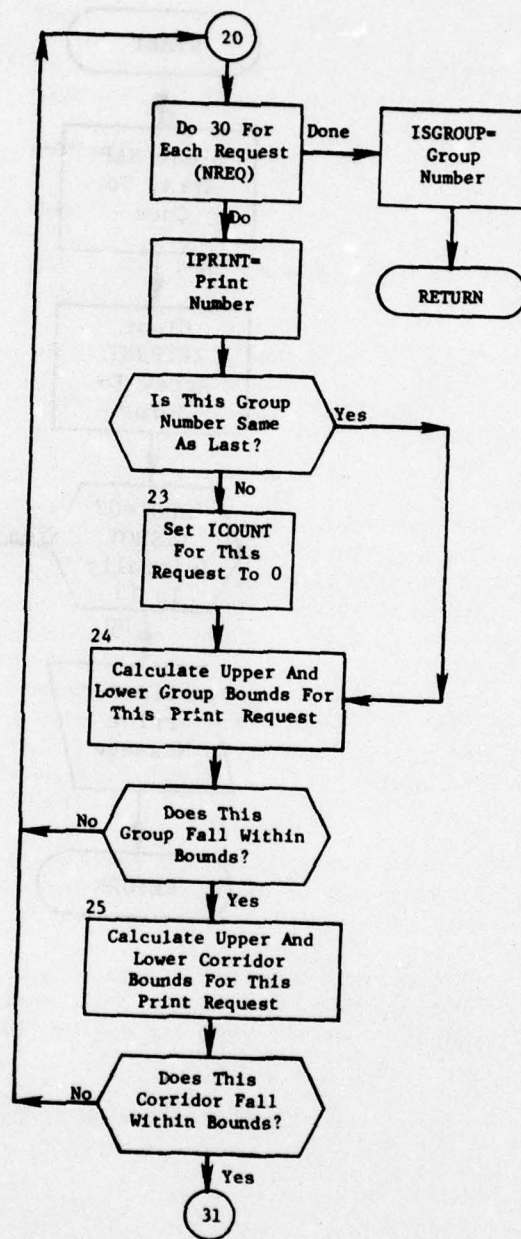


Figure 66. (Part 2 of 3)

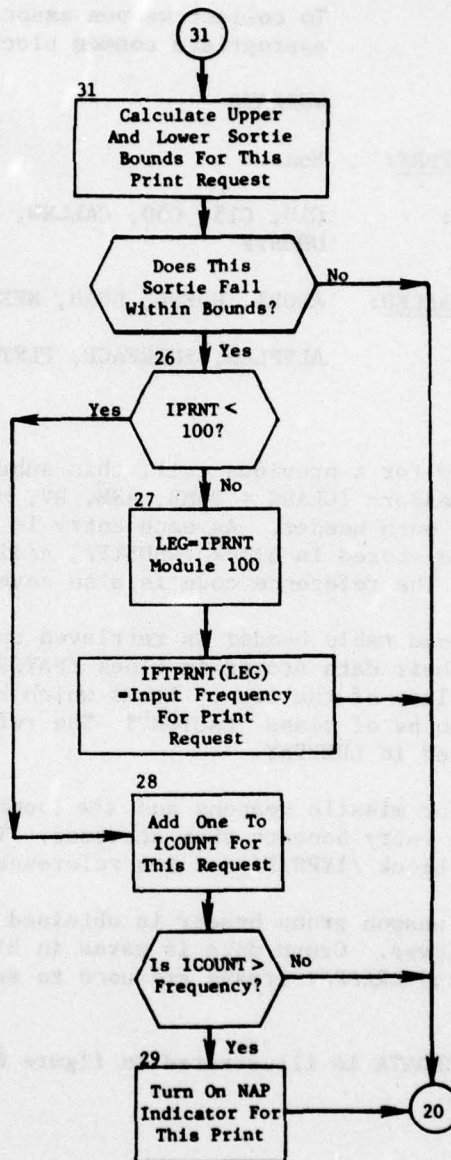


Figure 66. (Part 3 of 3)

4.7.5 Subroutine WEpdata

PURPOSE: To collect weapon associated data and store it in appropriate common blocks

ENTRY POINTS: WEpdata

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, CALLSW, GRPSTF, PAYSTF, TYPSTF, WHDSTF

SUBROUTINES CALLED: ABORT, HDFND, HEAD, NEXTTT, RETRV

CALLED BY: ALTPLAN, INTRFACE, PLNTPLAN

Method:

After checking for a previous call, this subroutine retrieves each of the warhead headers (CLASS = BOMB, ASM, RV, MRV and MIRV) and all entries beneath each header. As each entry is retrieved its data, (YIELD and FFRAC) are stored in block /WHDSTF/, additional data is stored when CLASS = ASM. The reference code is also saved in array LREFWHD.

Next the payload table header is retrieved and each payload table is retrieved and their data stored in block /PAYSTF/. LREFWHD is used to determine the class of the item. Items which have no match for LREFWHD are assumed to be of class "FACTOR". The reference code of the payload table is stored in LREFPAY.

The headers for missile weapons and the bomber weapons are now retrieved and each type entry beneath them accessed. The data from each type entry is stored in block /TYPSTF/ and its reference code saved in LREFTYP.

Finally, the weapon group header is obtained and each weapon group record is retrieved. Group data is saved in block /GRPSTF/. In addition, the LREFTYP and LREFPAY arrays are used to set type and payload indexes respectively.

Subroutine WEpdata is illustrated in figure 67.

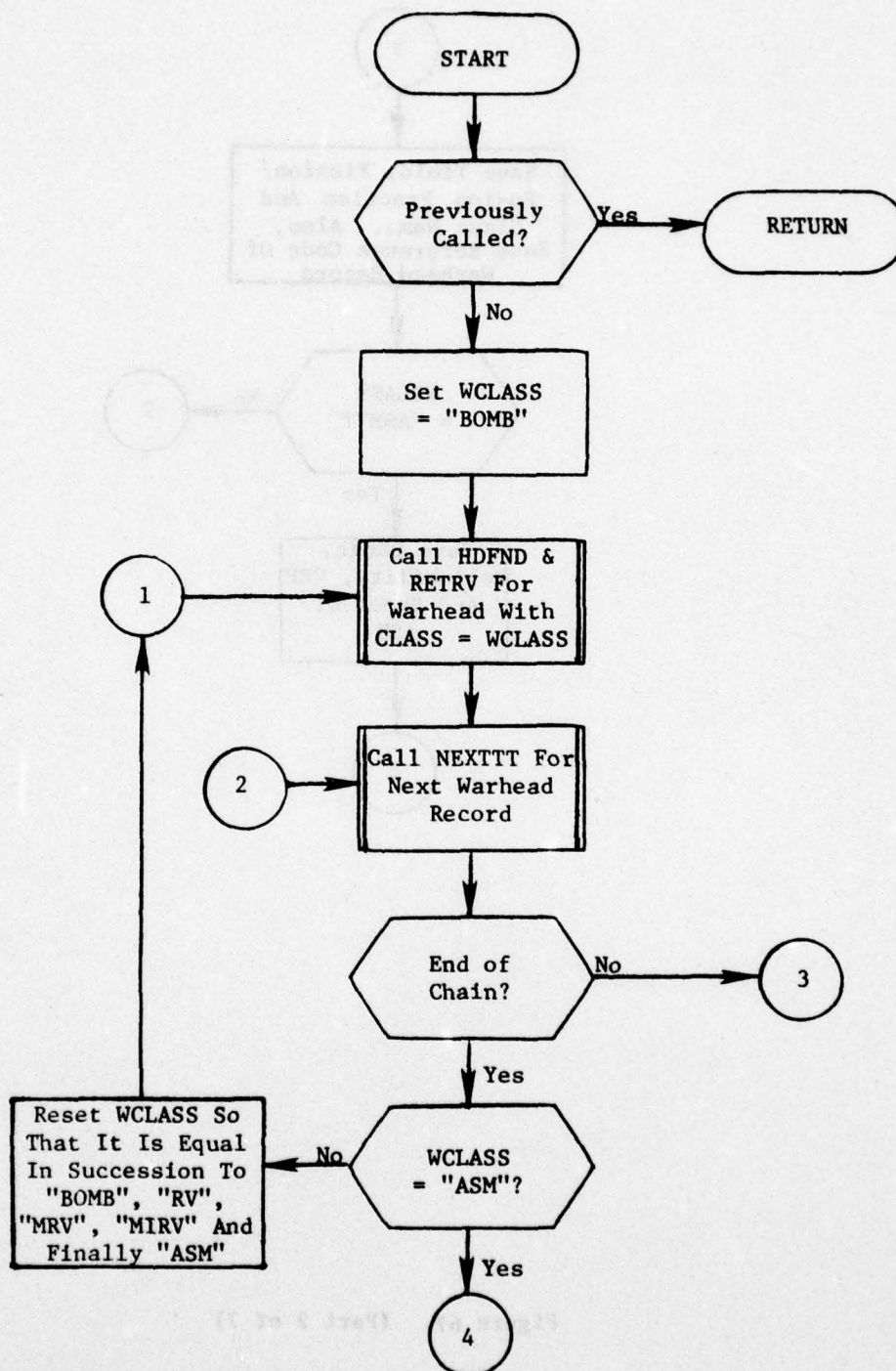


Figure 67. Subroutine WEPDATA (Part 1 of 7)

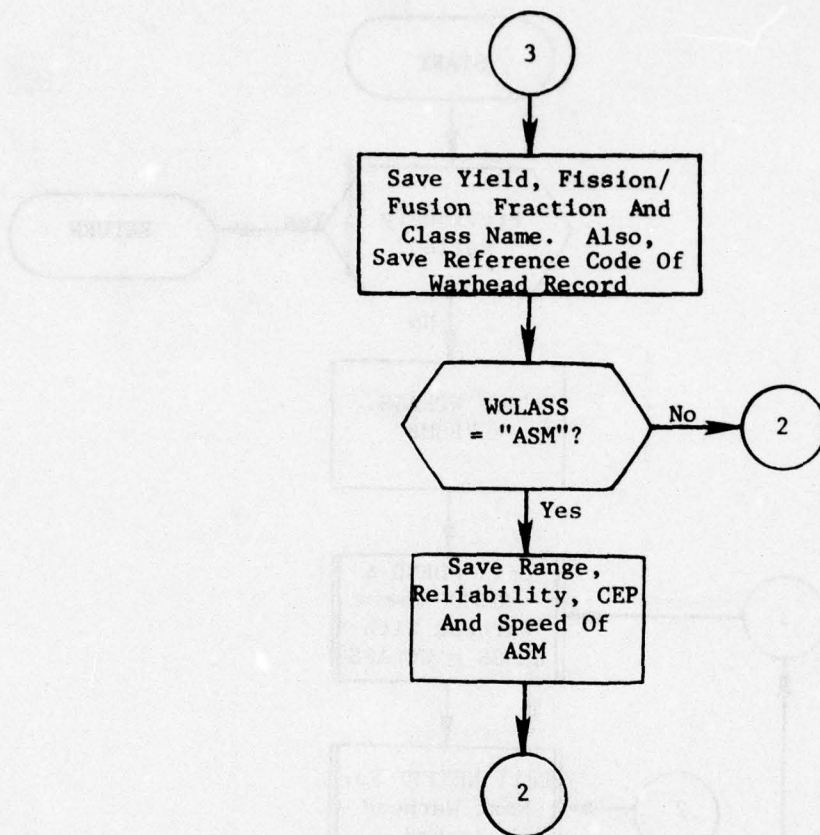


Figure 67. (Part 2 of 7)

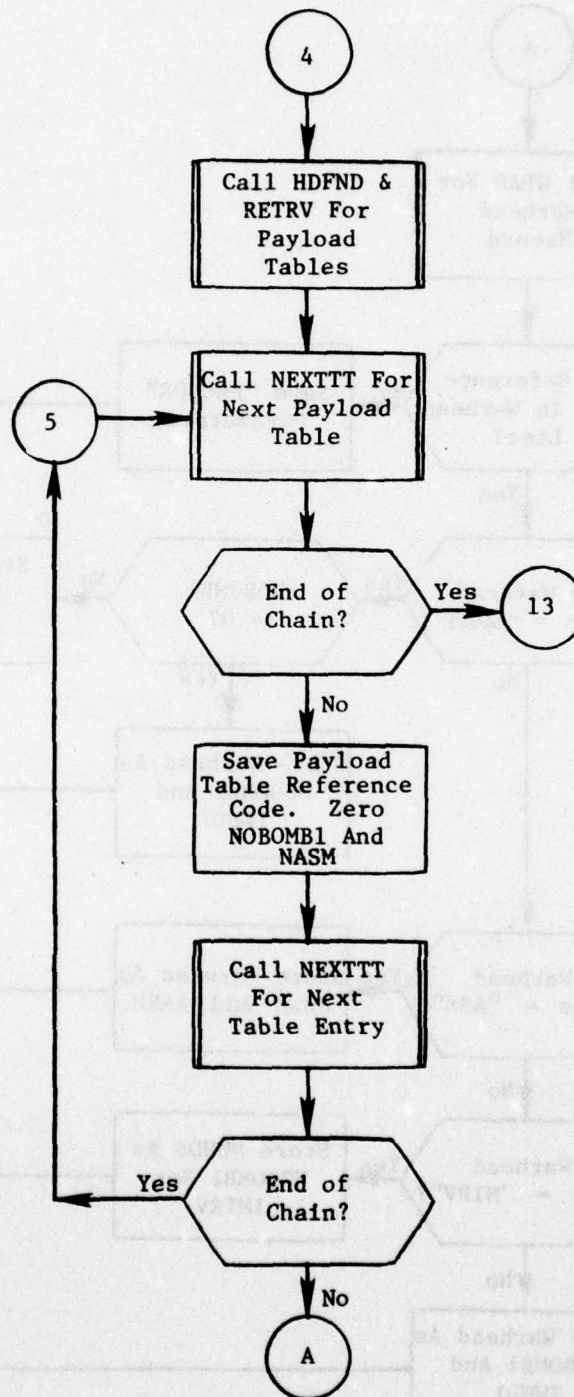


Figure 67. (Part 3 of 7)

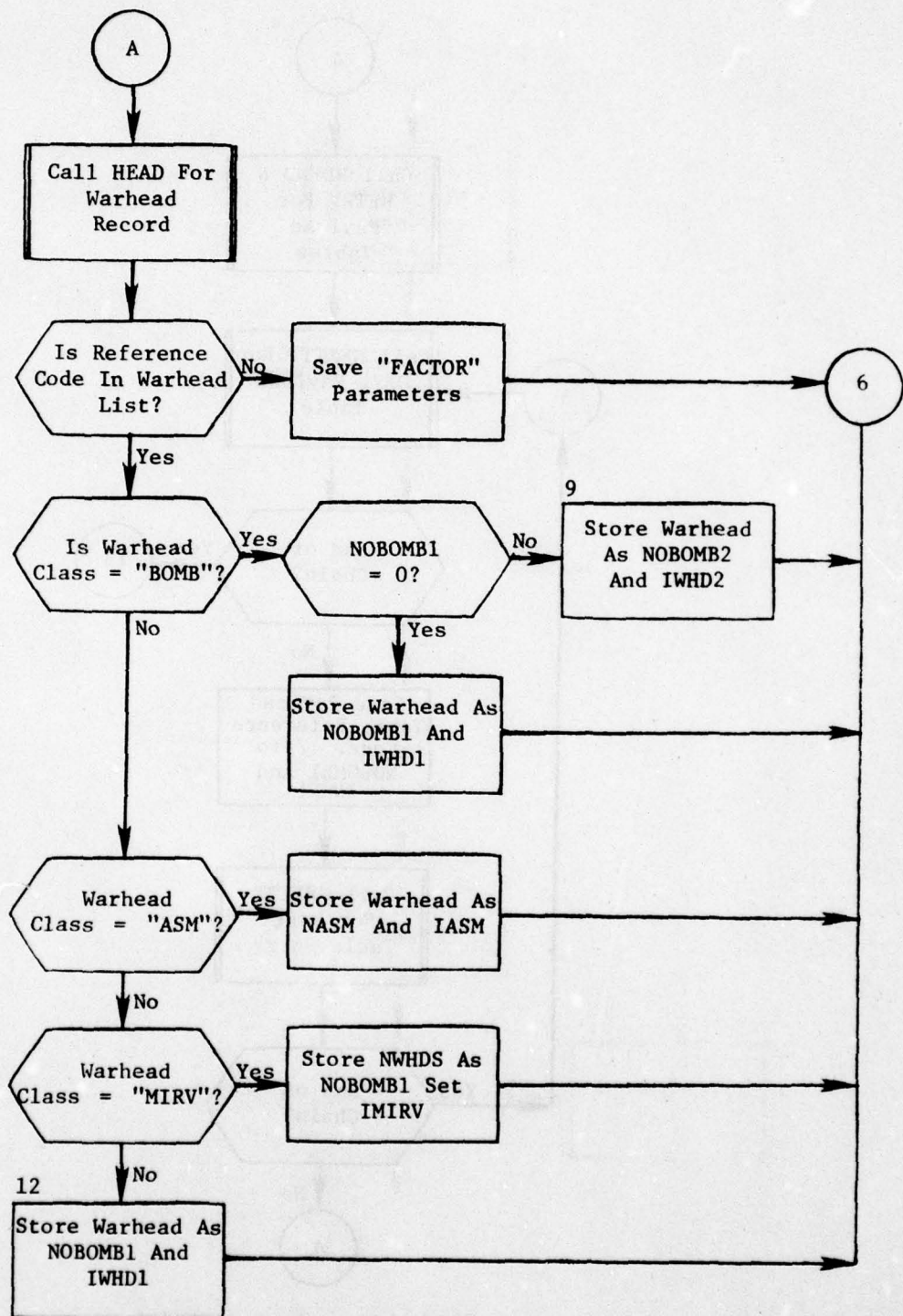


Figure 67. (Part 4 of 7)

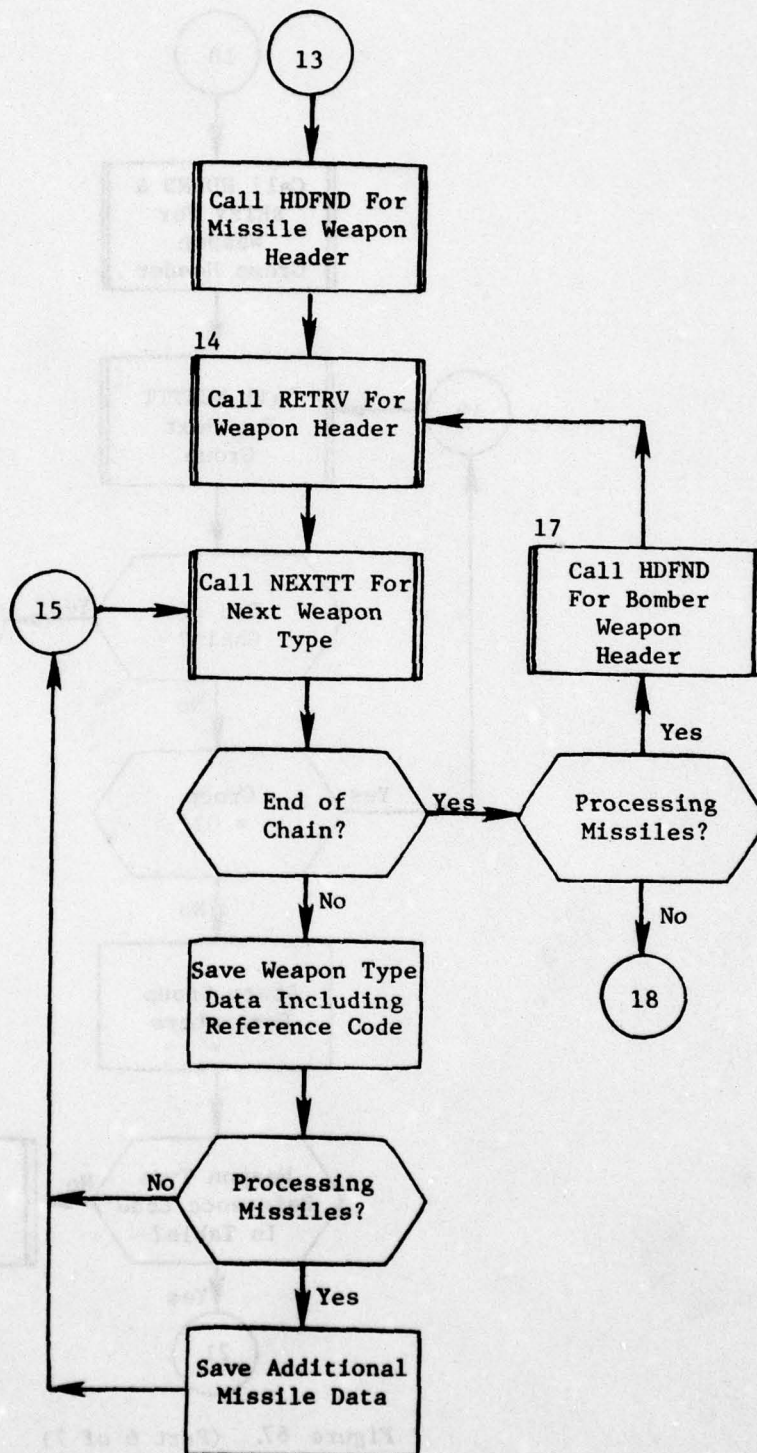


Figure 67. (Part 5 of 7)

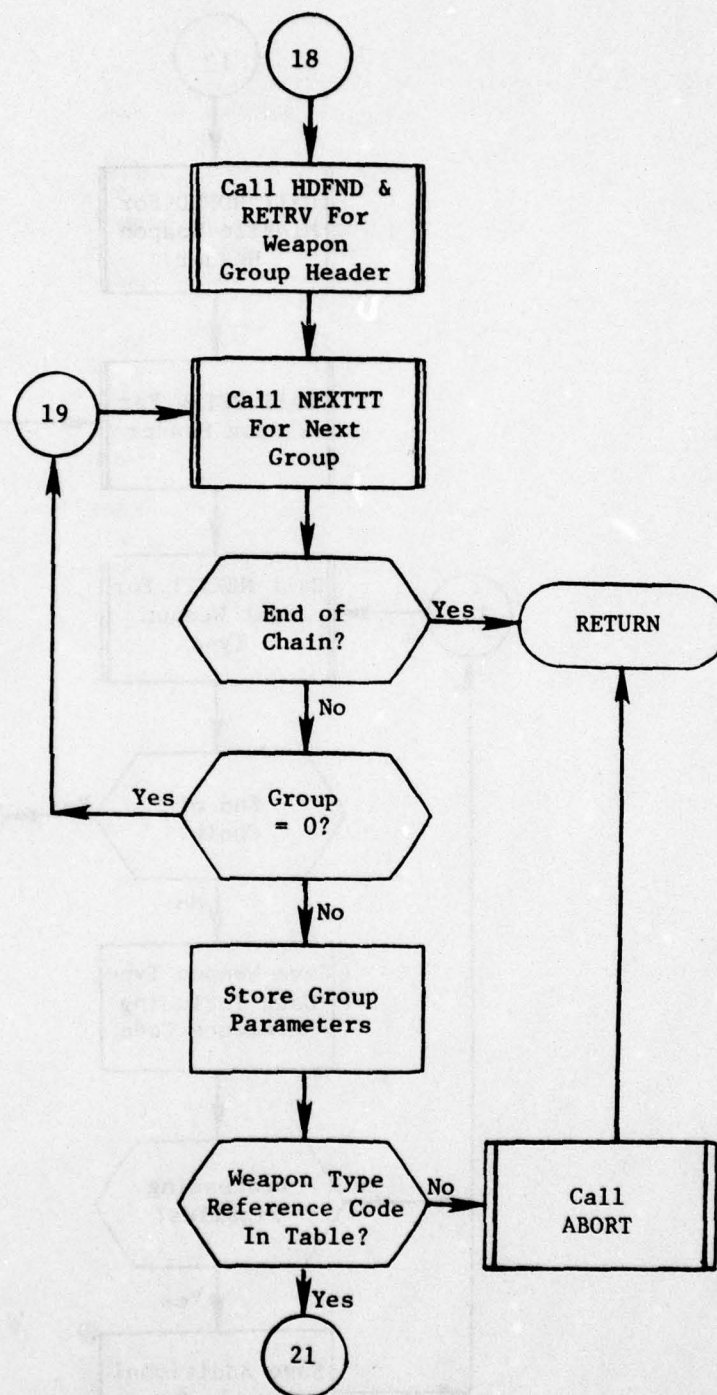


Figure 67. (Part 6 of 7)

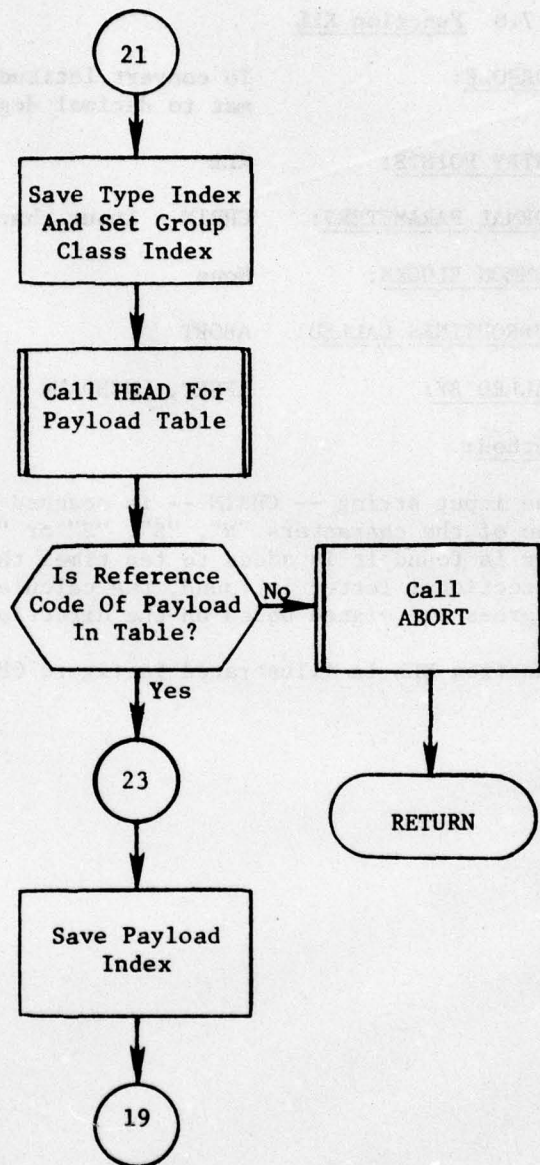


Figure 67. (Part 7 of 7)

4.7.6 Function XLL

PURPOSE: To convert latitude or longitude from DDMMSS format to decimal degrees.

ENTRY POINTS: XLL

FORMAL PARAMETERS: CHRIN - Input character string

COMMON BLOCKS: None

SUBROUTINES CALLED: ABORT

CALLED BY: IFSET, LNCHDATA

Method:

The input string -- CHRIN -- is scanned one character at a time until one of the characters "N", "S", "E" or "W" is found. Each time a number is found it is added to ten times the previous total. When the directional letter is found, the calculated total is converted to decimal degrees and signed based on the directional letter.

Function XLL is illustrated in figure 68.

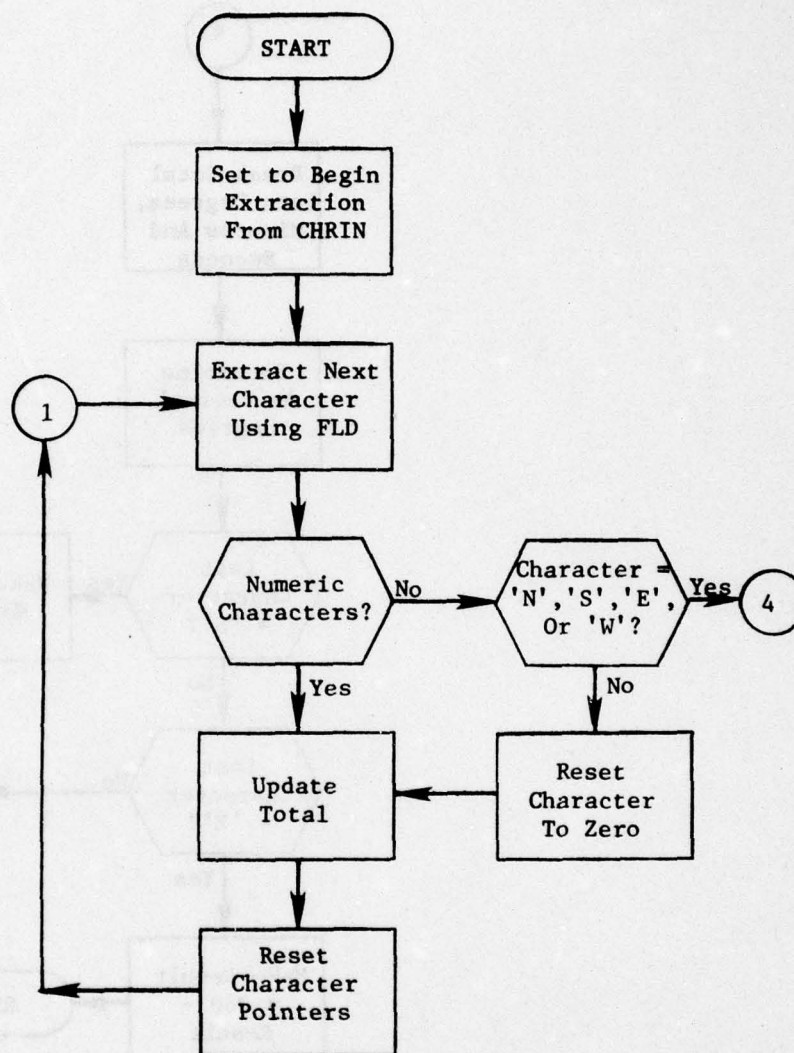


Figure 68. Function XLL (Part 1 of 2)

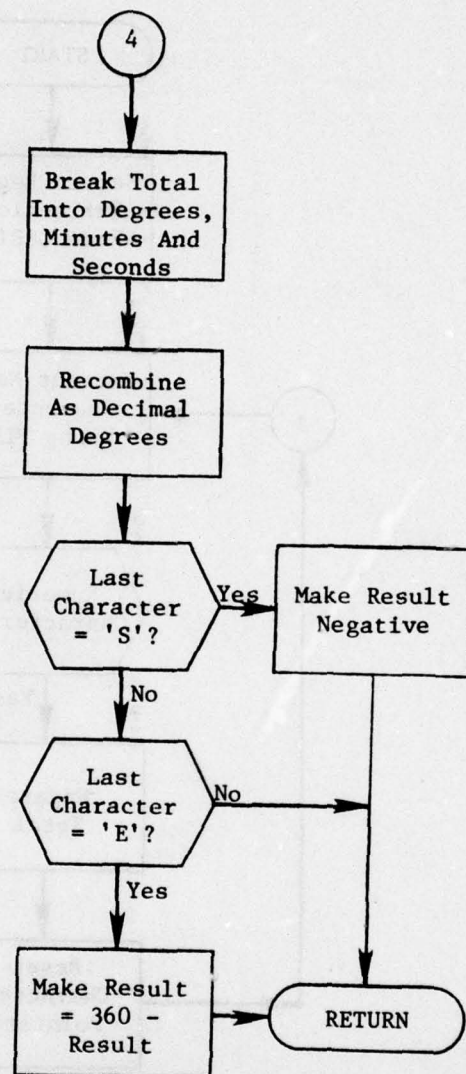


Figure 68. (Part 2 of 2)

4.8 Subroutine PLNTPLAN*

PURPOSE: Driver subroutine for overlay which performs the sortie completion function

ENTRY POINTS: PLNTPLAN

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, C10, C15, C30, CONTROL, GRPSTF, IRF, LASREF, MISCT, OUTSRT, PAYSTF, PPINFO, RECBAS, RECOVERY, TANKA

SUBROUTINES CALLED: DIRECT, GEOGET, HDFND, HEAD, ITLE, KERPLUNK, LNCHDATA, NEXTTT, PLANBOMB, PLANTMIS, RETRV, SLOG, SNAPCON, SNAPIT, SORBOMB, WEpdata

CALLED BY: ENTMOD (PLANOUT)

Method:

After some initialization, WEpdata and GEOGET are called for basic data retrieval. The refuel area and recovery base arrays are now setup. LNCHDATA is called and the tanker data is printed. The sortie header is retrieved to begin the main process. Each sortie is read in and then processed. PLANTMIS is called to process missile sorties and PLANBOMB is called to process bomber sorties. Prior to calling PLANBOMB, the bomber sortie is checked to see if it is a "one-way" sortie or a sortie in which the bomber is not fully loaded. After the call to PLANBOMB, SORBOMB creates the completed sortie in the integrated data base.

When all sorties have been processed, the lists of "one-way" and "not fully utilized" sorties are displayed along with a table of recovery base utilization.

Subroutine PLNTPLAN is illustrated in figure 69.

* First subroutine of overlay PLNT -- this overlay performs both sortie completion and sortie change functions. PLNTPLAN is the sortie completion function driver.

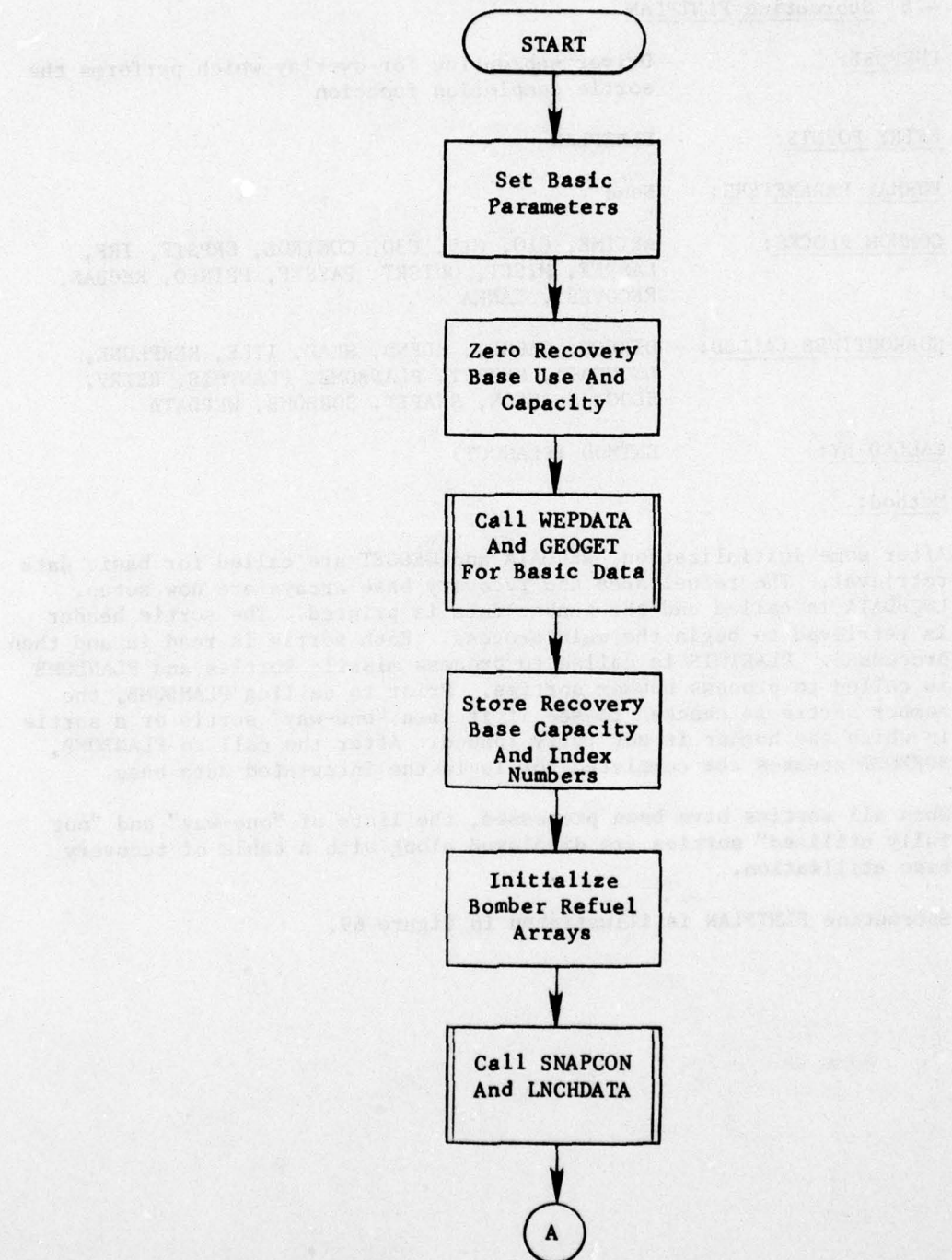


Figure 69. Subroutine PLNTPLAN (Part 1 of 7)

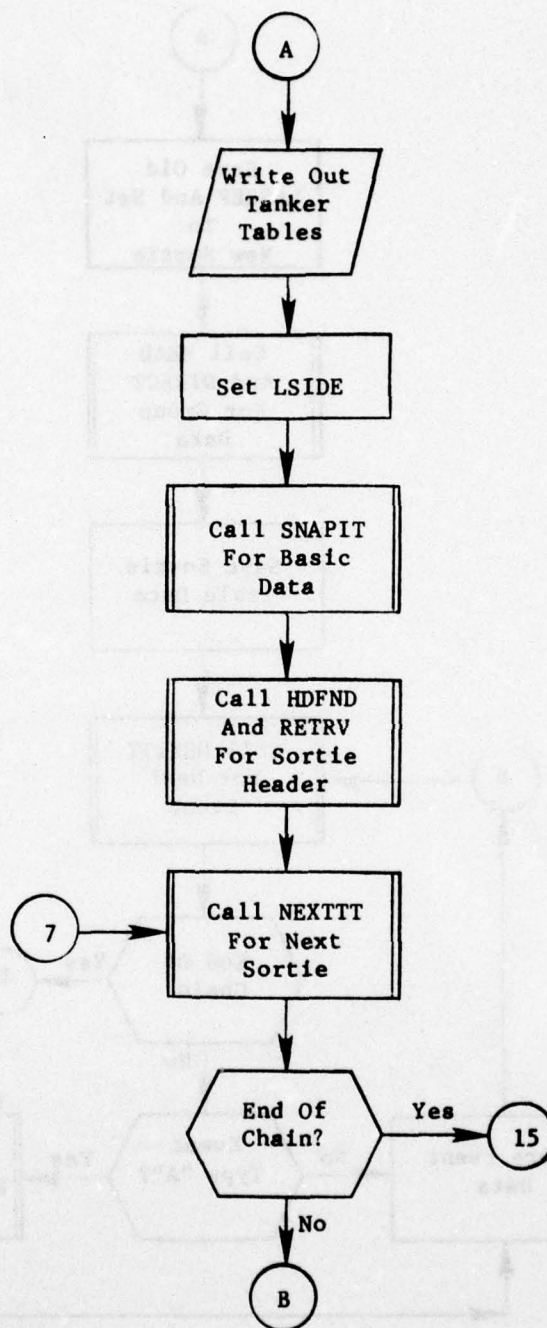


Figure 69. (Part 2 of 7)

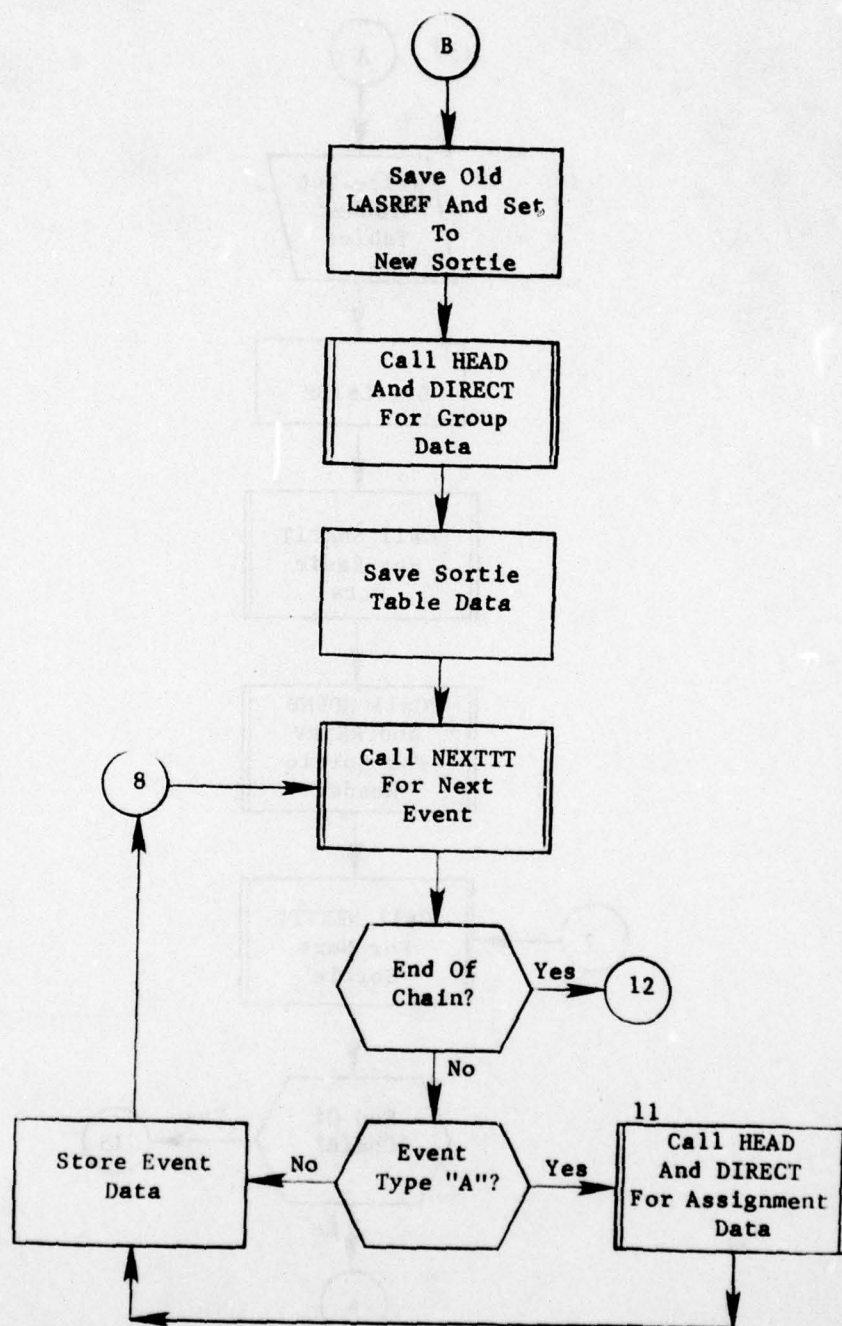


Figure 69. (Part 3 of 7)

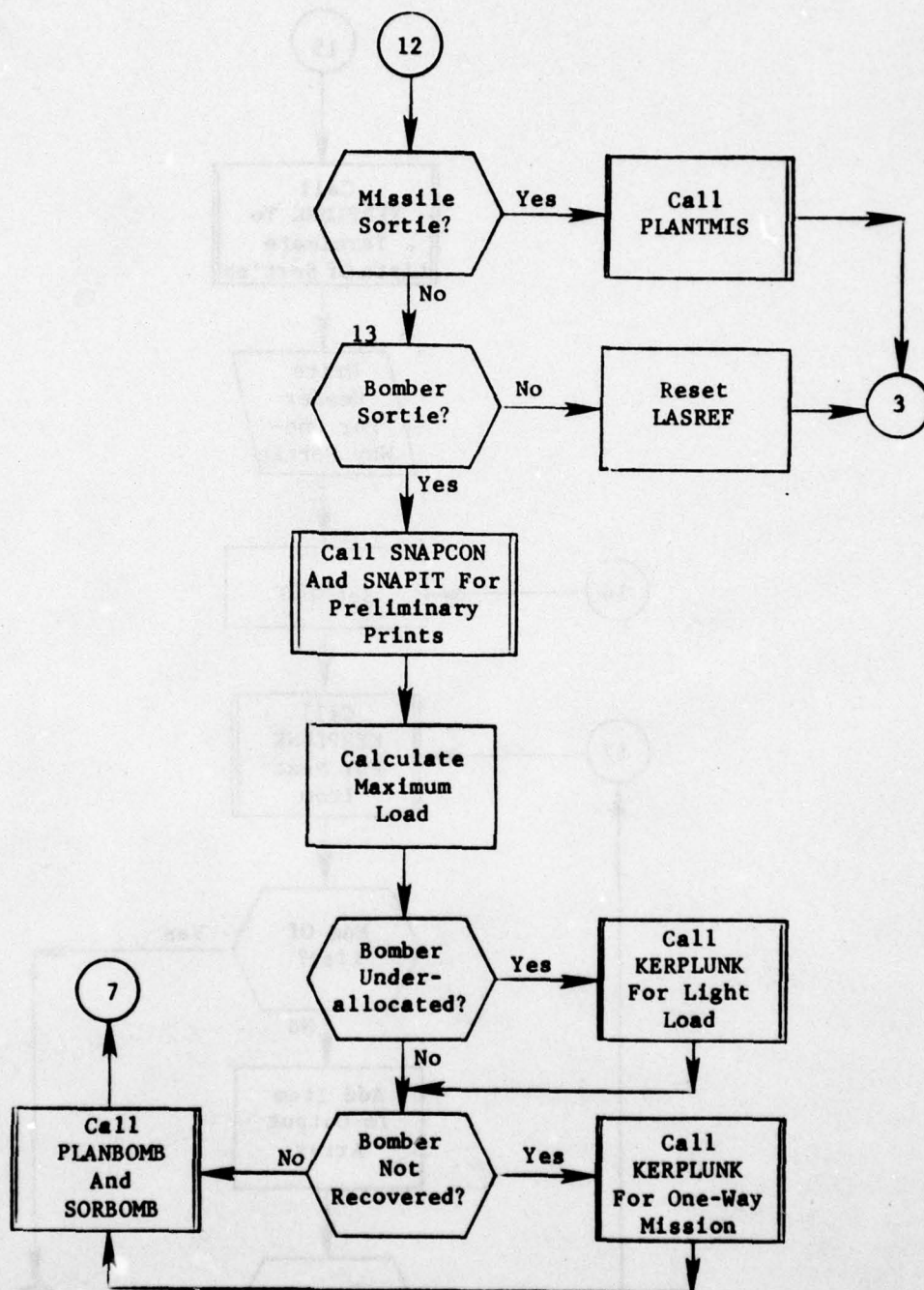


Figure 69. (Part 4 of 7)

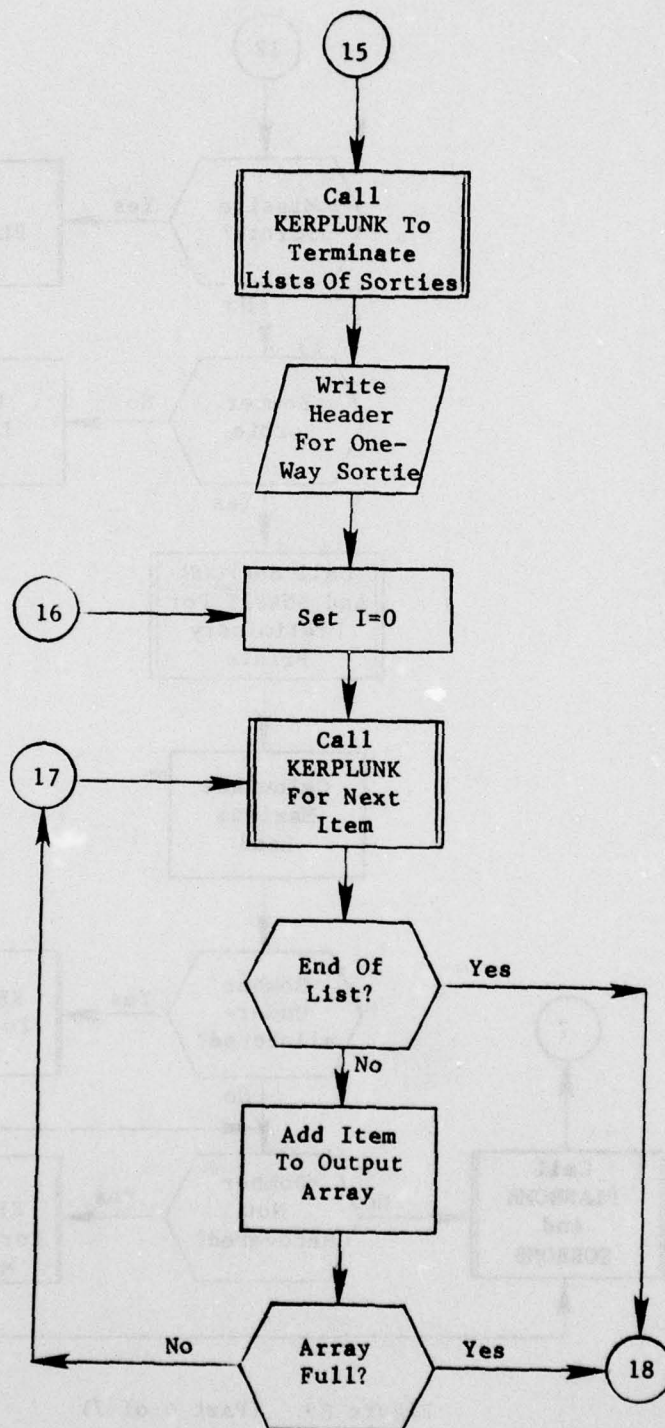


Figure 69. (Part 5 of 7)

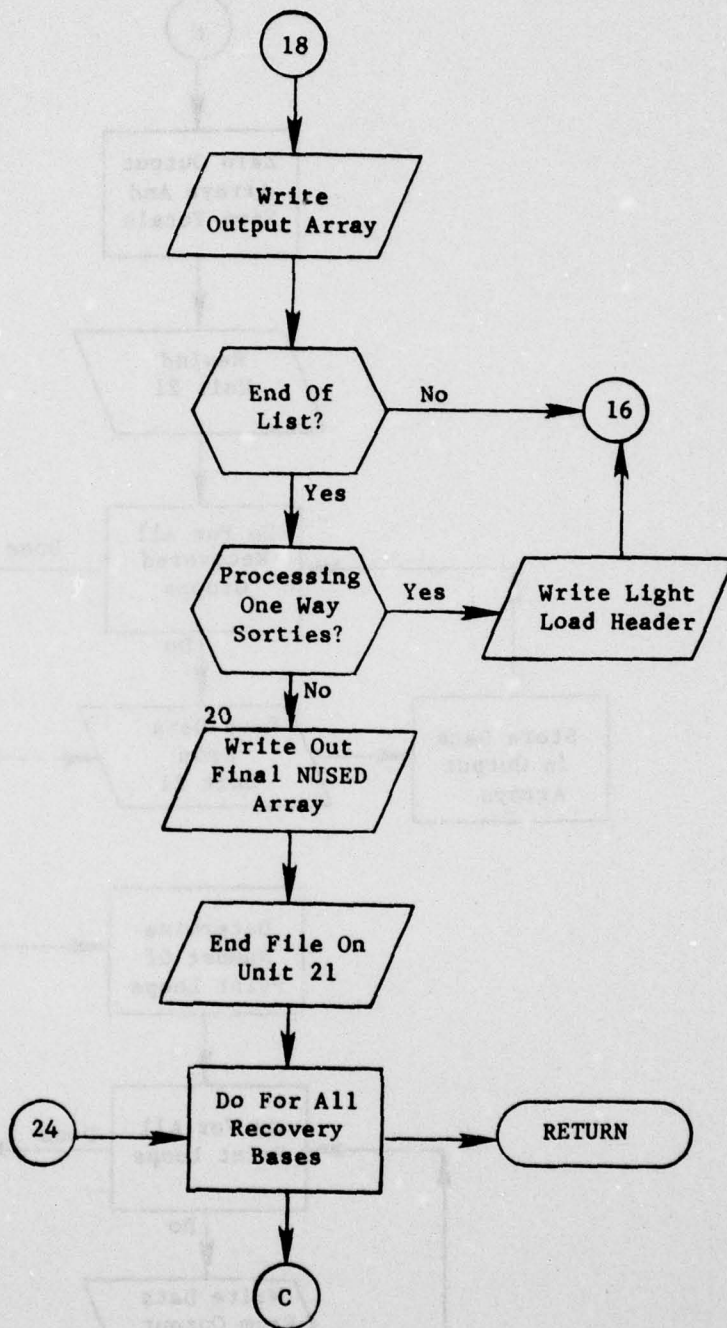


Figure 69. (Part 6 of 7)

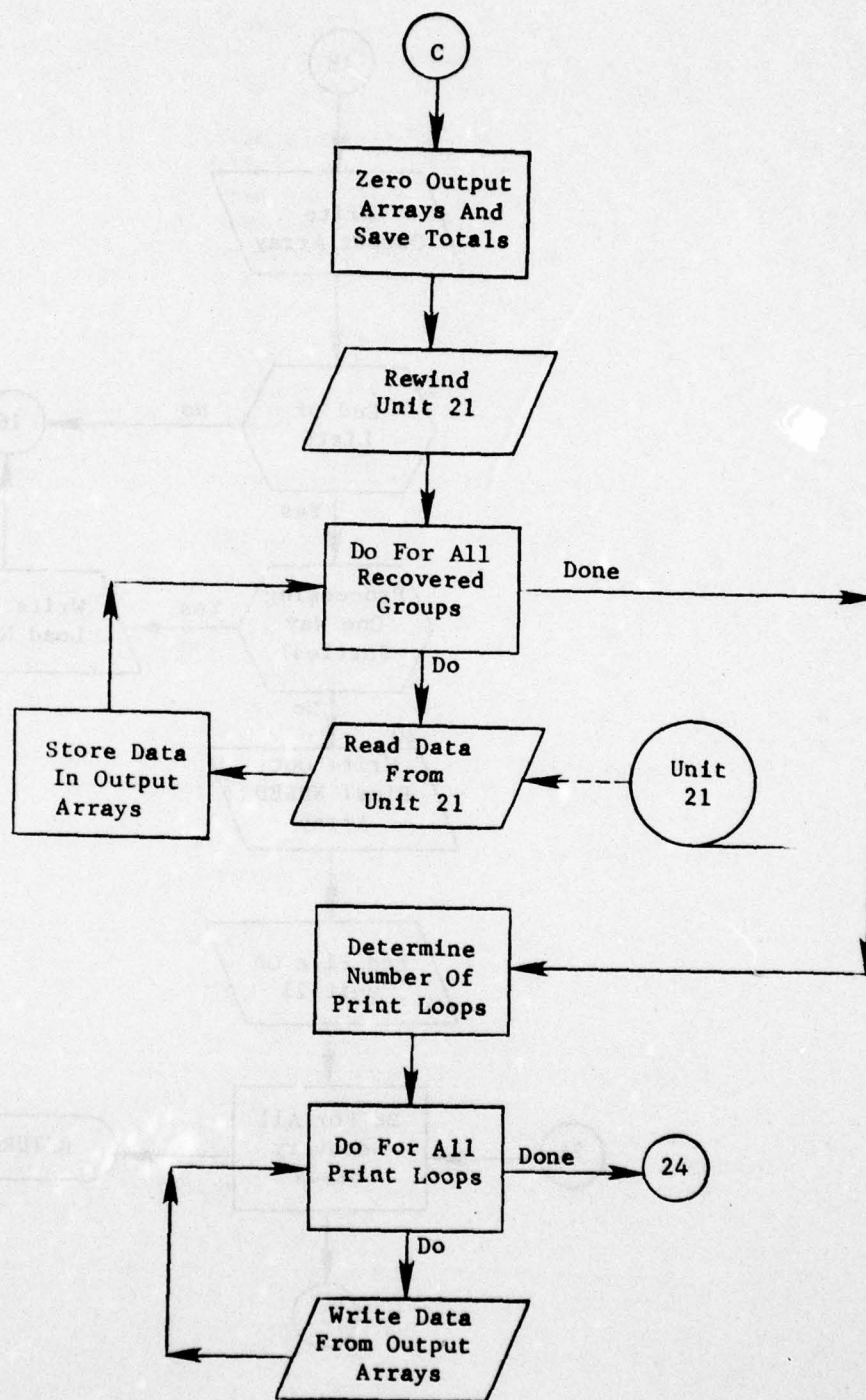


Figure 69. (Part 7 of 7)

4.8.1 Subroutine ALTPLAN*

PURPOSE: Driver for sortie change function.

ENTRY POINTS: ALTPLAN

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, C10, C30, CORRCL, CORRCHAR, ERCOM, GRPSTF, LASREF, OOPS, OUTSRT

SUBROUTINES CALLED: DATIM, DIRECT, DISTF, DLETE, FINDME, GEOGET, HEAD, INSGET, INSPUT, LNCHDATA, MODFY, NEXTTT, ORDER, PLANBOMB, PLANTMIS, RETRV, SLOG, SNAPCON, SORBOM, STORE, TOFM, WEPDATA

CALLED BY: ENTMOD (PLANOUT)

Method:

User Inputs

The user may alter any sortie developed within the QUICK system. Subroutine ALTPLAN, reads these sortie change requests and updates the data base accordingly. Three adverbs are recognized and are:

- 'CCARD' -- indicates change strike
- 'ICARD' -- indicates insert strike
- 'ACARD' -- indicates add sortie

The CCARD Clause. The general form of the CCARD (see UM 9-77, Volume IV for complete discussion) is:

CCARD sortie number, desig1, desig2, [, hob, dec, rac, tchange]
[* caloff, dlatoff, dlongoff]

The first three parameters specify the action to be performed and must be entered for each adverb. The "sortie number" indicates which sortie is to be changed. Various modes of entries for the target DESIG's are:

- o "desig1" will be dropped when "desig2" is blank (that is, a comma appears in lieu of a target DESIG)
- o Strikes are replaced when both "desig1" and "desig2" are non-blank and not equal. "desig1" will be replaced by "desig2" (and if a complex, it must be the representative target)

* Second subroutine of overlay PLNT. This overlay performs both sortie completion and sortie change functions. The sortie change function driver is ALTPLAN.

- o When "desig1" equals "desig2", elements of the strike are changed. This allows a change in down time, height of burst, offset characteristics or depenetration corridor.

Following "desig2" parameters that may be changed are optional. These options come in two collections. In each collection the individual parameters may be omitted but their preceding commas must still appear. The first collection contains the options of changing the height-of-burst, specifying a new depenetration corridor, suppressing recalculation of attrition, and altering the flight time. The second collection permits the definition of target offset. This collection must be introduced by the asterisk (*) operator. Also, if no options are used from the first collection, default commas are not required. Similar logic applies if the fourth, or third and fourth, or second, third or fourth options are not employed.

The ICARD Clause. The general form of the clause is

$$\begin{aligned} & \text{ICARD } \underline{\text{sortie number}} , \left[\underline{\text{desig1}} \right] , \underline{\text{desig2}} \left[, \underline{\text{hob}} , \underline{\text{dec}} , \underline{\text{rac}} , \right] \\ & \quad \left[\underline{\text{tchange}} \right] \\ & \quad \left[* \underline{\text{caloff}} , \underline{\text{dlatoff}} , \underline{\text{dlongoff}} \right] \end{aligned}$$

"desig2" will be inserted after "desig1". If "desig1" is omitted (two commas after the sortie number), "desig2" will become the first target of the sortie. The discussion of optional information for the CCARD clause on new targets applies to "desig2".

The ACARD Clause. This clause is used to add nonMIRV missile sorties. This clause has the general form:

$$\begin{aligned} & \text{ACARD } \underline{\text{desig}} , \underline{\text{hob}} , \underline{\text{group}} , \underline{\text{siteind}} , \left[, \underline{\text{isal}} , \underline{\text{tlaun}} \right] \\ & \quad \left[* \underline{\text{caloff}} , \underline{\text{dlatoff}} , \underline{\text{dlongoff}} \right] \end{aligned}$$

Generally, all comments concerning the CCARD clause apply to the ACARD clause. Note, that no sequence number is supplied; PLANOUT will supply the correct value. "group" is the weapon group number containing the launch base. "siteind" is the INDEXNO of the site from the weapon group. If no launch time (tlaun) is given, the program sets this time according to salvo number and launch interval. If simultaneous launches are desired, salvo numbers ("isal") must be repeated for each round which is to be salvoed; i.e., if SIMLAUNCH is i, the missile salvo number j would be repeated i times in order to have i weapons launch at (j-i)*(launch interval). In the case of nonsalvoed missiles and bombers, launch will occur at the earliest feasible time as determined by alert status, CORMSL, etc. If a launch time is specified, that value is added to the delay times discussed above.

Processing within ALTPLAN closely follows user requests and is described below.

General

This subroutine is best followed by reference to figure 70 as much of the logic is involved in data checking of input clauses. After preliminary calls to obtain the necessary data for execution and to FINDME which retrieves desired group, sortie and target records, the subroutine begins to process the input clauses one at a time.

ACARD Processing

If the clause encountered is an ACARD clause, the subroutine checks the clause for errors. In the process of this check the desired target and group records are retrieved by FINDME. The last nontanker sortie is now retrieved. The range to the target is checked for system limits and the time of flight, salvo, and launch time calculated. The values for the new ASSIGN (record type 70) record are stored in /C30/ and STORE is called. Finally, the values for the SRTEVA (record type 50) record are stored in /C30/ and STORE is called. When all record storage is complete, PLANTMIS is called to complete the sortie. Then all sorties which occur after the new sortie (i.e., tanker sorties) are retrieved and modified with their sortie numbers incremented.

CCARD and ICARD Processing

If the clause encountered in processing is a CCARD or ICARD clause, all remaining clauses are checked to see if they involve the same sortie. All such clauses (all those with the same sortie number) are then processed before any clauses involving other sorties are processed.

When the first set of change clauses (CCARD and/or ICARD) for any sortie are processed, the contents of common block /CORRC1/ are calculated. This block contains information on defended areas of penetration corridors for subroutine FLTSORT.

Each applicable clause is error checked in turn. In the process FINDME retrieves the appropriate sortie table (SRTYTB). Furthermore, unless the first input DESIG value is blank, the TARCDE record corresponding to the target identified by the first input DESIG value is retrieved and the first attendant weapon assignment (ASSIGN) from the proper group found. The remainder of the process depends upon whether the clause was a CCARD or ICARD clause and the values of the two DESIGs input (DES1 and DES2).

In the case of a CCARD the second DESIG(DES2) is checked. If DES2 is blank, the ASSIGN record retrieval above is deleted. If DES1 equals DES2, the subroutine checks on which data items have been modified and then the ASSIGN, SRTEVA and SRTYTB records are modified as required. If DES1 is not equal DES2 and DES2 is a legal target DESIG, the old ASSIGN record is deleted and the process proceeds as in an ICARD below.

For an ICARD (or from a CCARD above), the process is to create a new ASSIGN and SRTEVA record in the proper chain order. This order is established by the value of DES1. If DES1 is blank, the new assignment appears as the first assignment event. In any case the target list record for DES2 is retrieved and the new ASSIGN and SRTEVA records stored as per the input clause.

When all clauses pertaining to a particular sortie have been processed, the subroutine carries out the completion process for that sortie as follows: If the sortie is a missile sortie PLANTMIS is called. In the case of a bomber sortie the process is more complex. The sortie is completely read into the /OUTSRT/ block. Then, unless specified otherwise by the user, the FLTSORT subroutine is called to recalculate flight time parameters. The PLANBOMB is called to complete the sorties and SORBOMB to update the integrated data base.

Subroutine ALTPLAN is illustrated in figure 70.

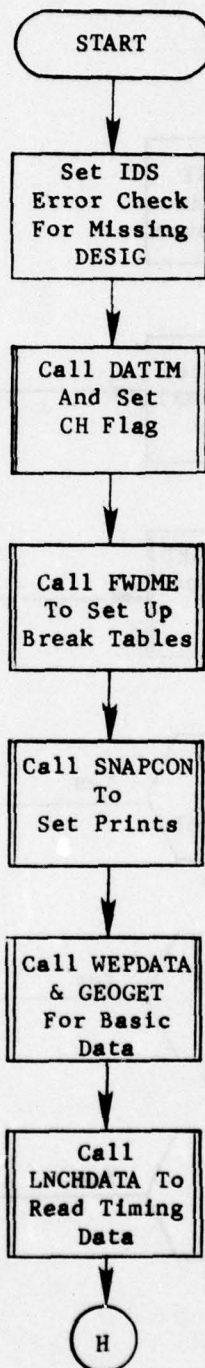


Figure 70. Subroutine ALTPLAN (Part 1 of 30)

AD-A058 406

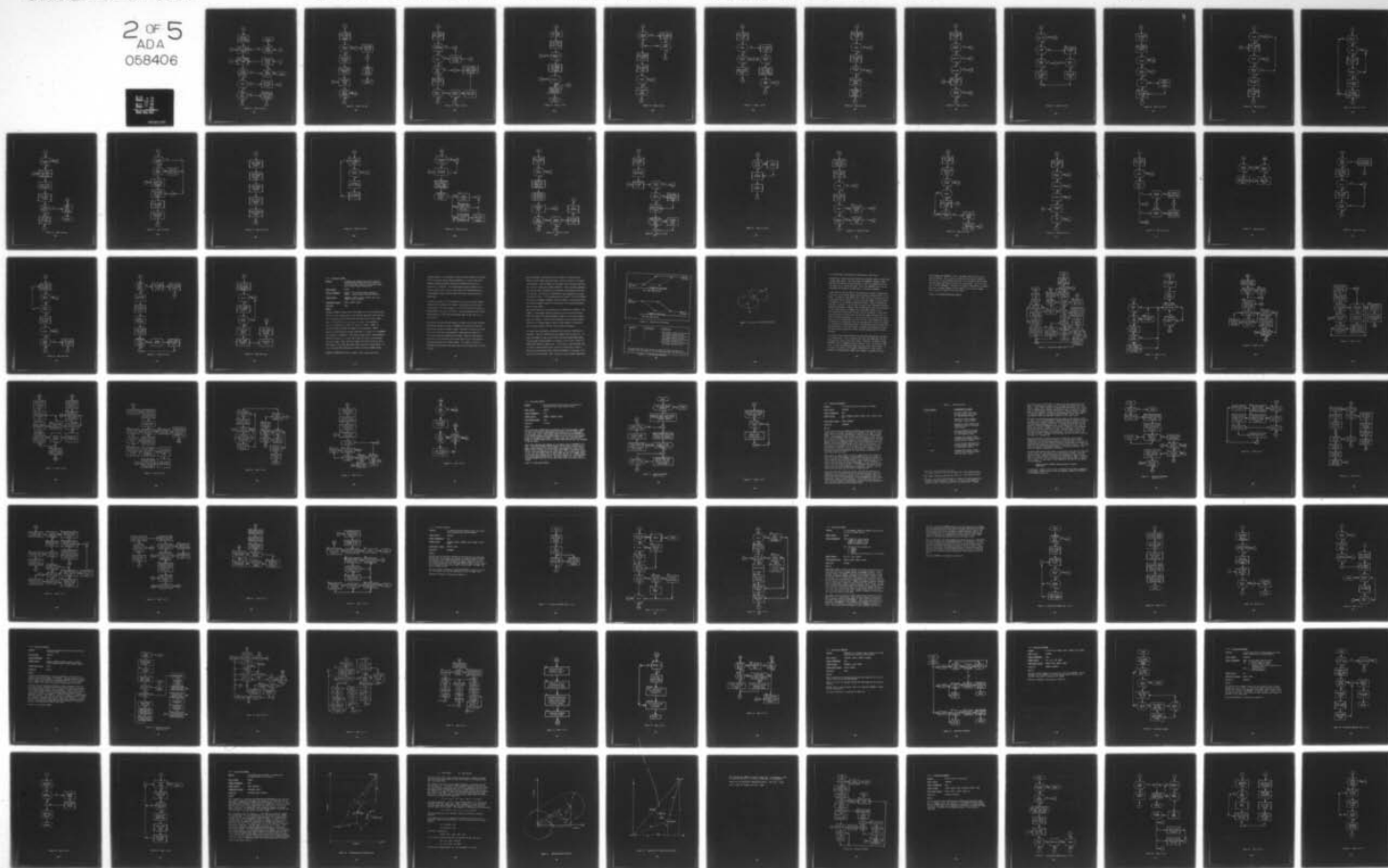
COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK). VOLU--ETC(U)
APR 78 D J SANDERS, P F MAYKRANTZ, J M HERRON
CCTC-CSM-MM-9-77-VOL-4-PT SBIE-AD-E100 085

F/G 15/7

UNCLASSIFIED

NL

2 OF 5
ADA
058406



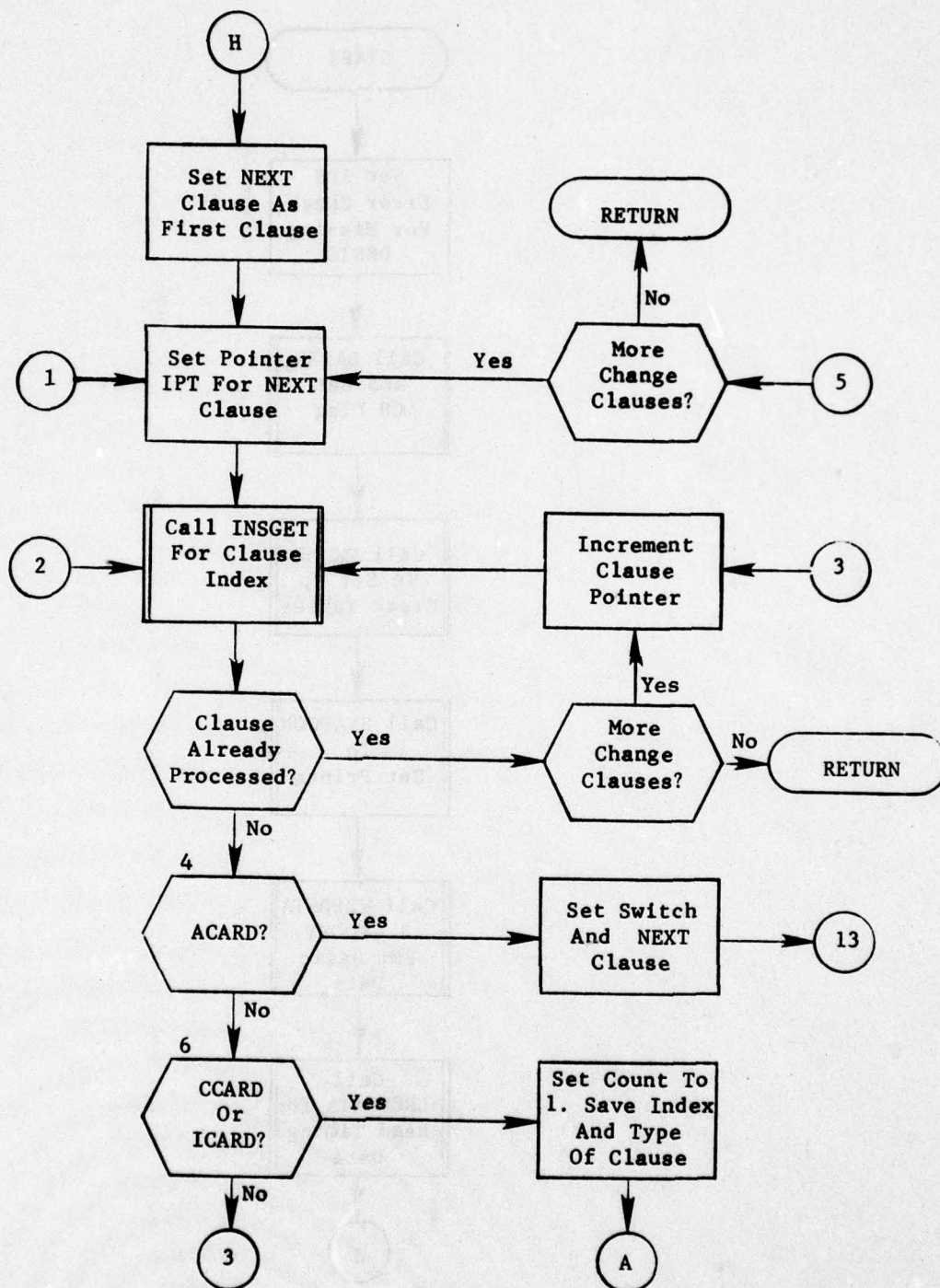


Figure 70. (Part 2 of 30)

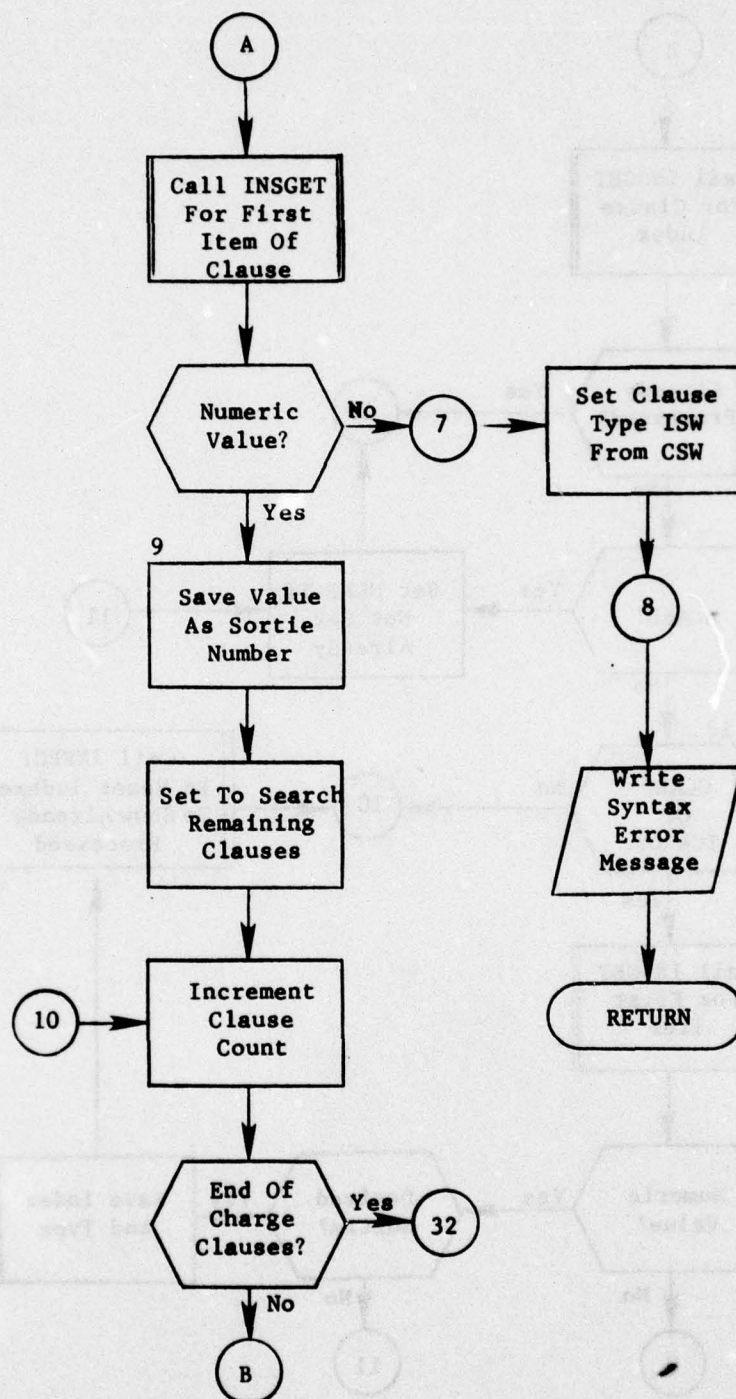


Figure 70. (Part 3 of 30)

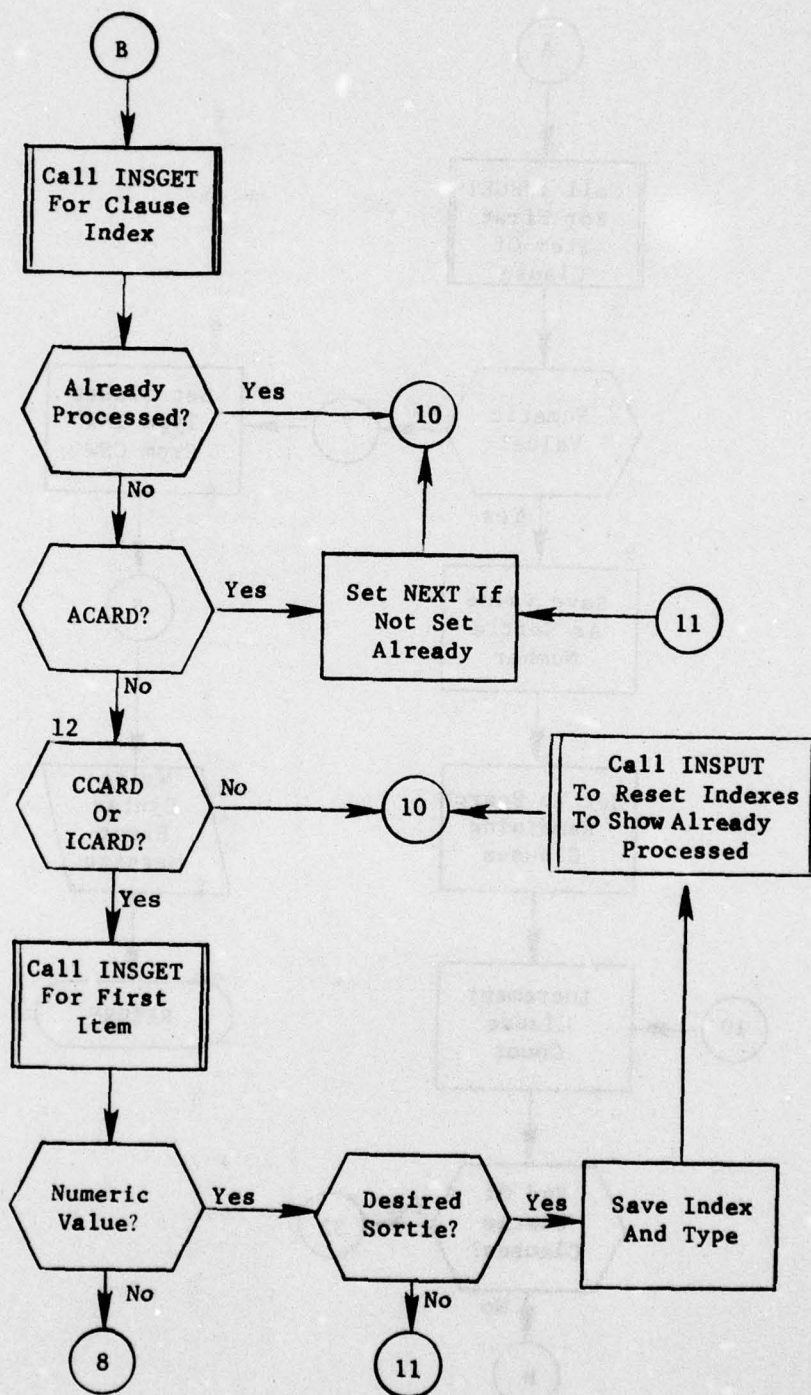


Figure 70. (Part 4 of 30)

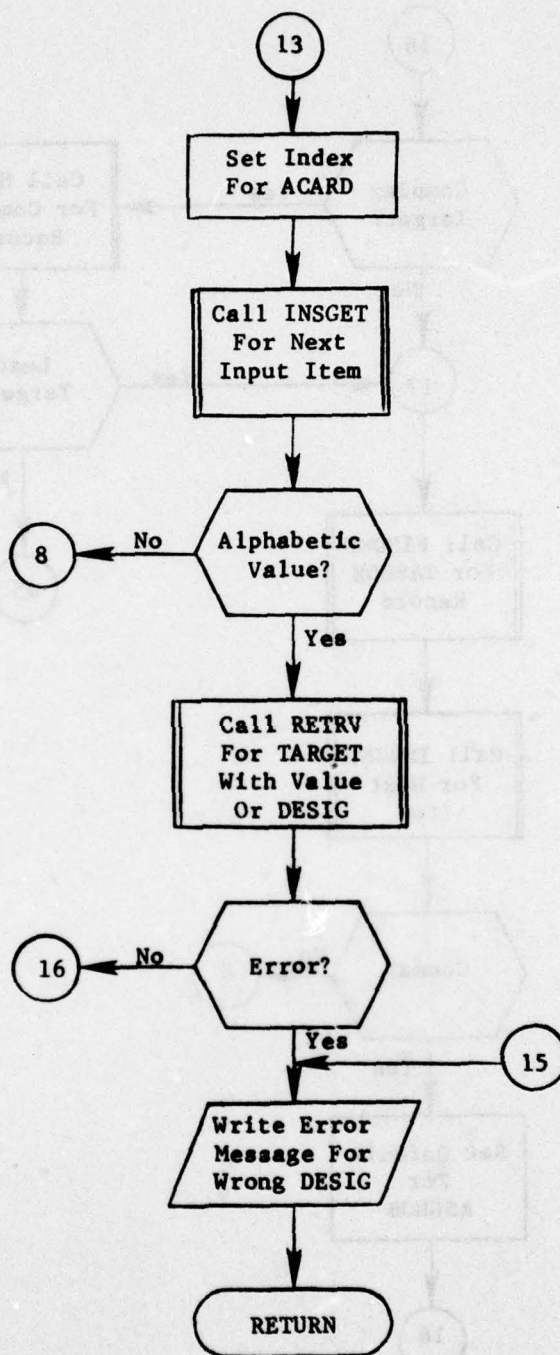


Figure 70. (Part 5 of 30)

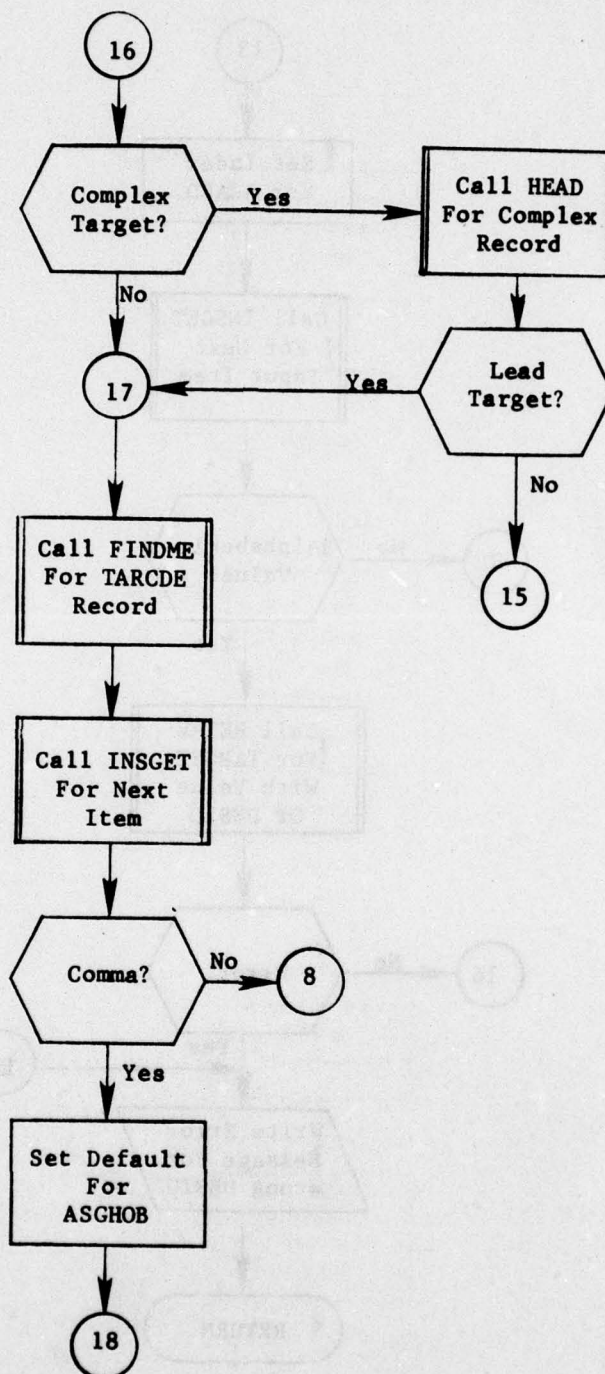


Figure 70. (Part 6 of 30)

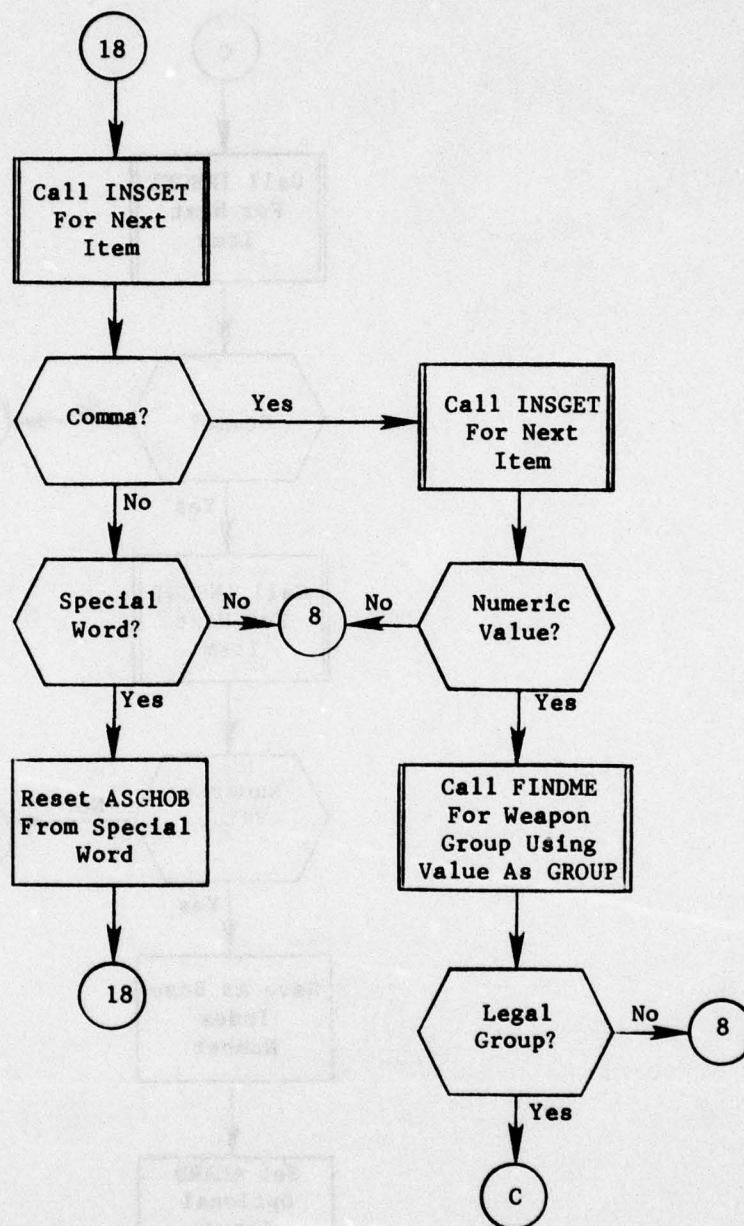


Figure 70. (Part 7 of 30)

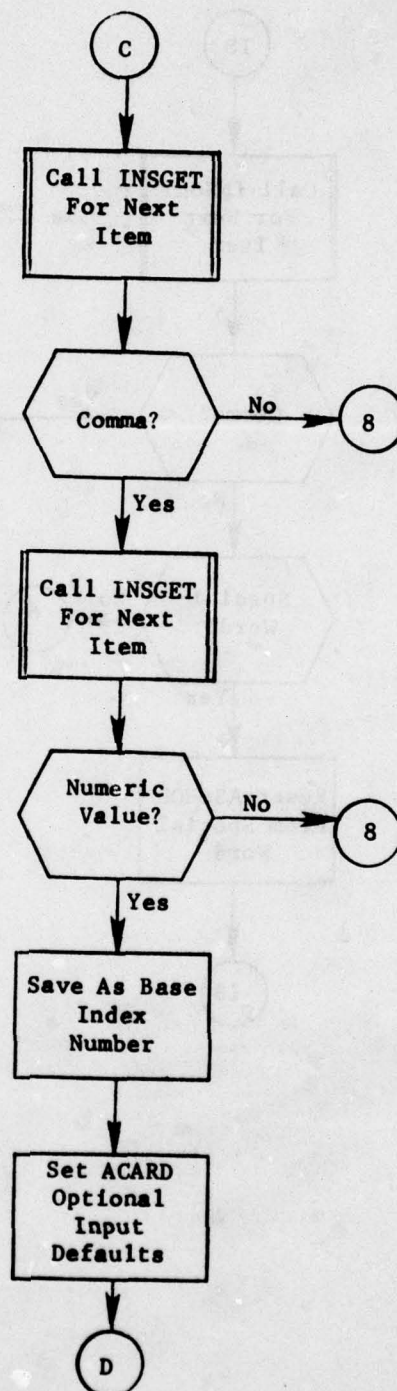


Figure 70. (Part 8 of 30)

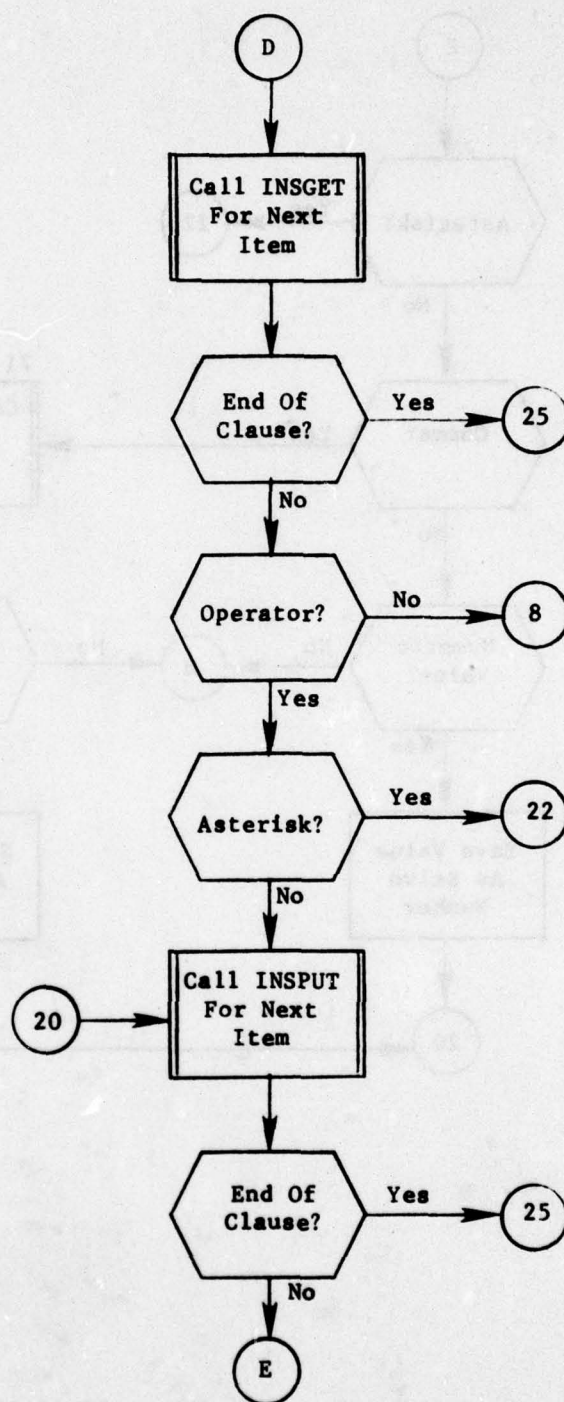


Figure 70. (Part 9 of 30)

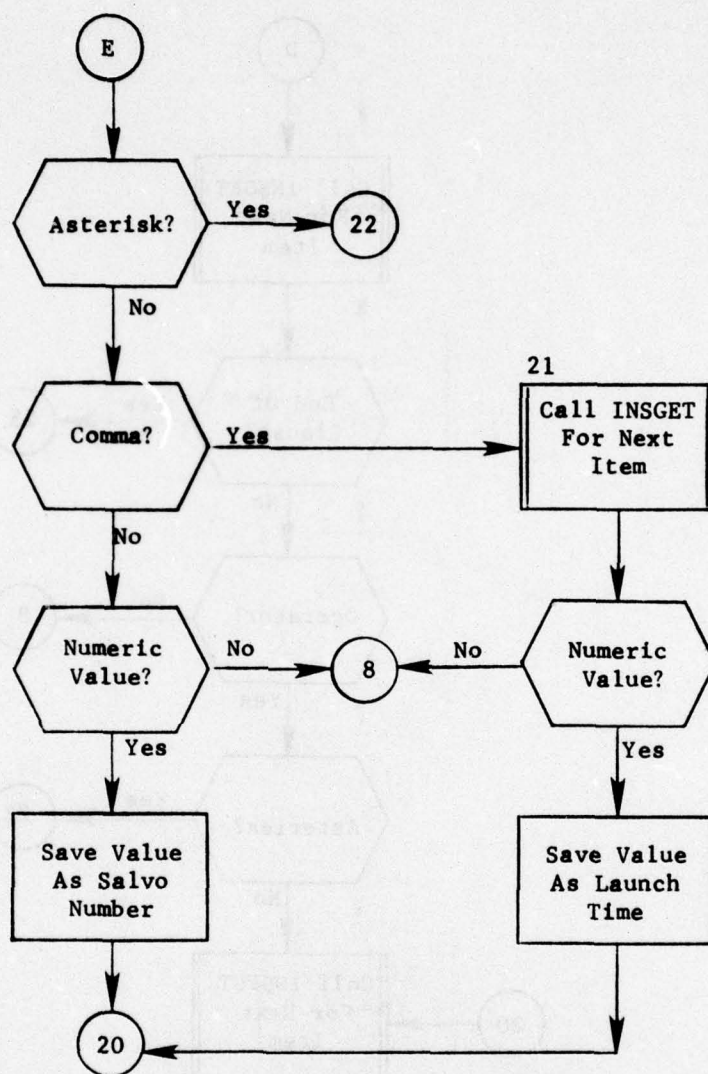


Figure 70. (Part 10 of 30)

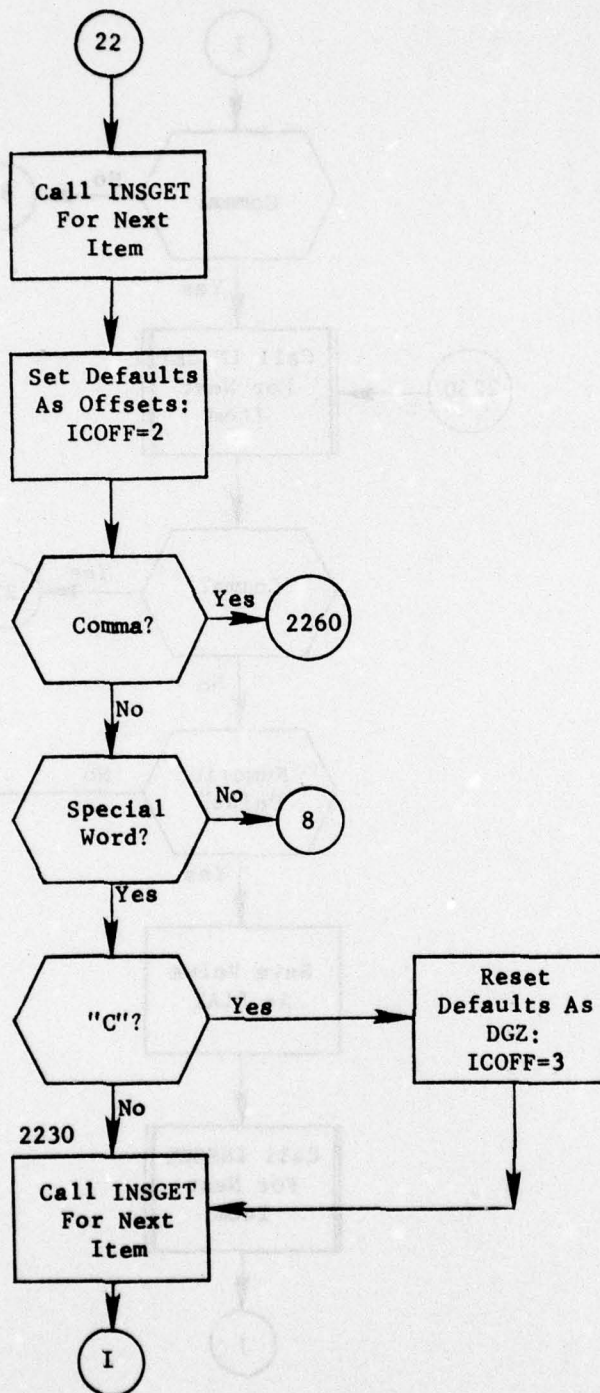


Figure 70. (Part 11 of 30)

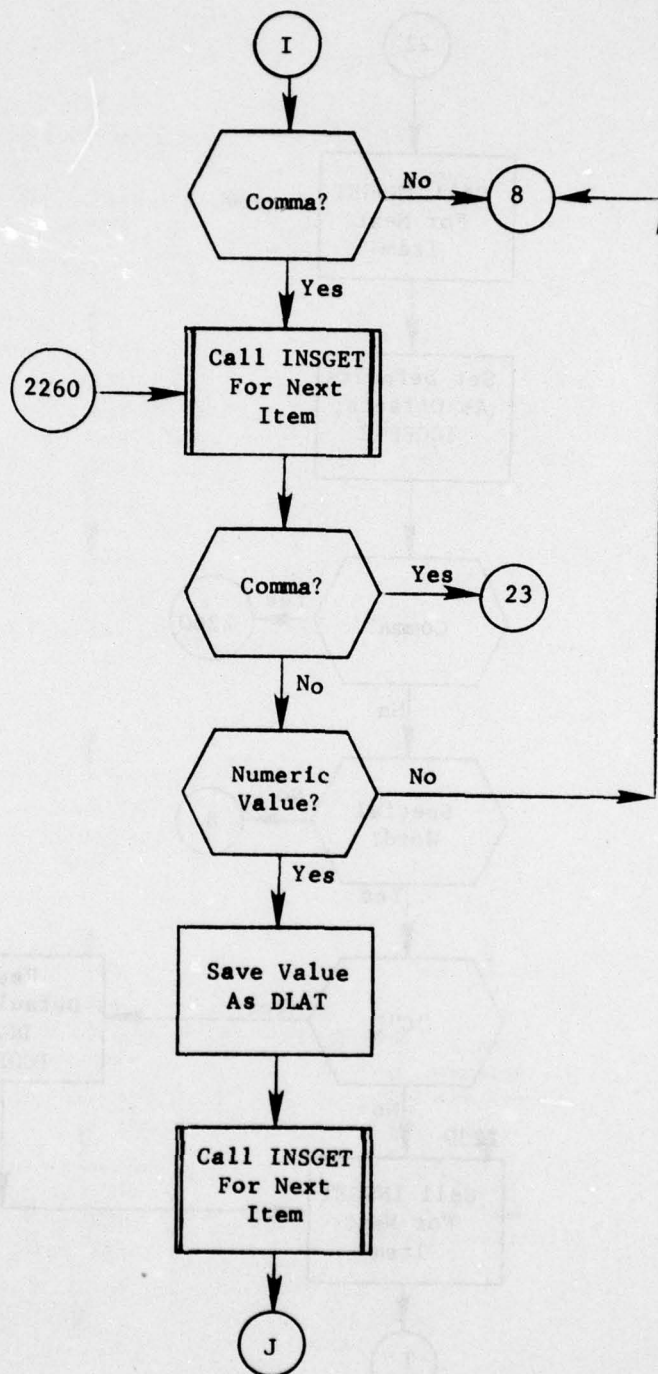


Figure 70. (Part 12 of 30)

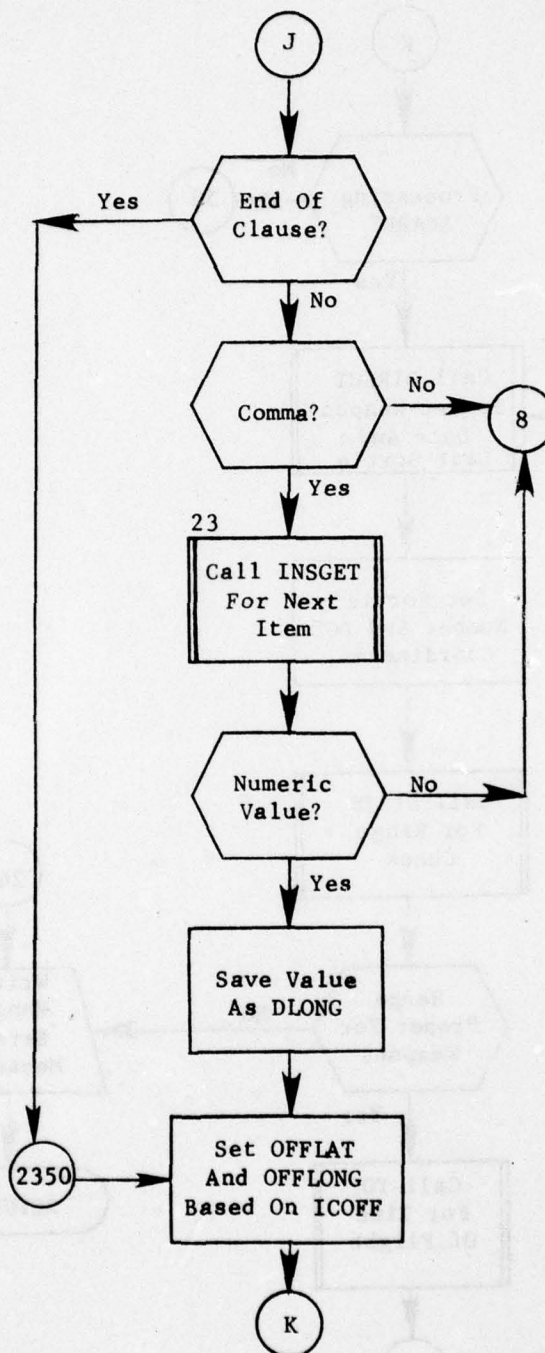


Figure 70. (Part 13 of 30)

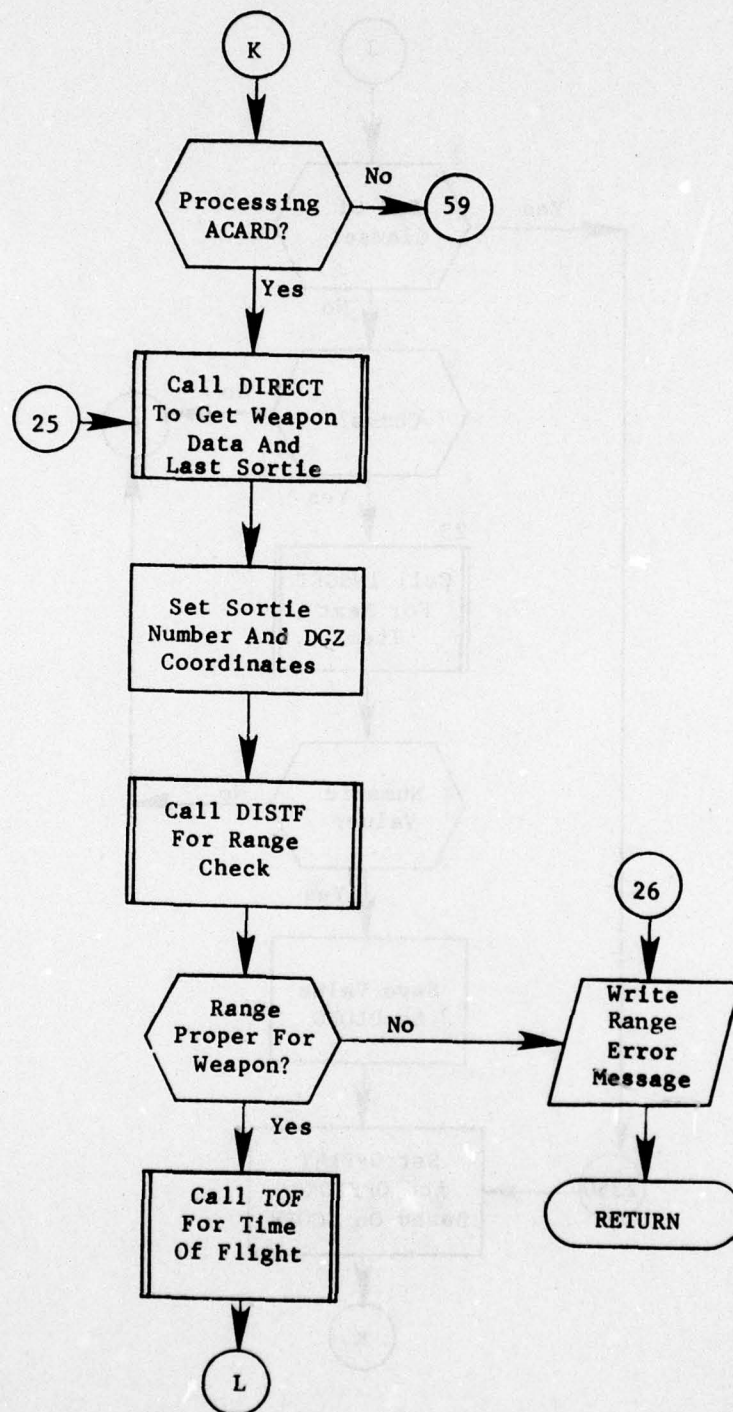


Figure 70. (Part 14 of 30)

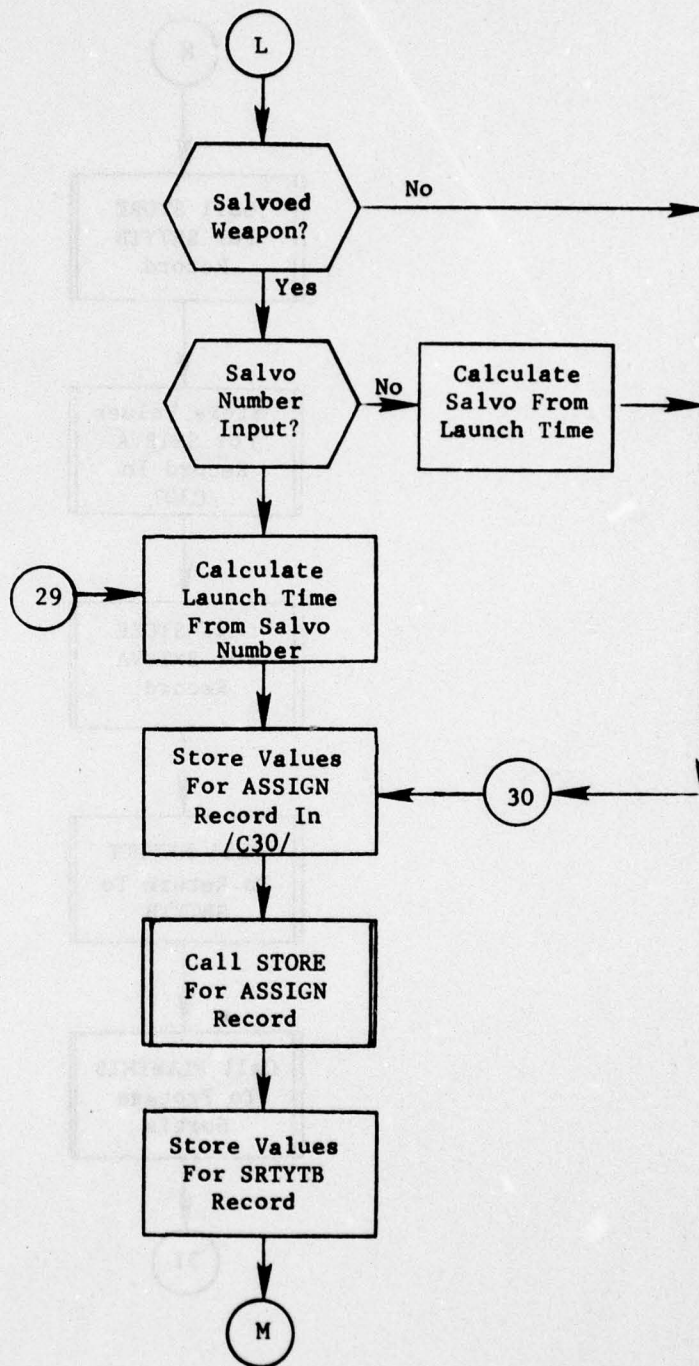


Figure 70. (Part 15 of 30)

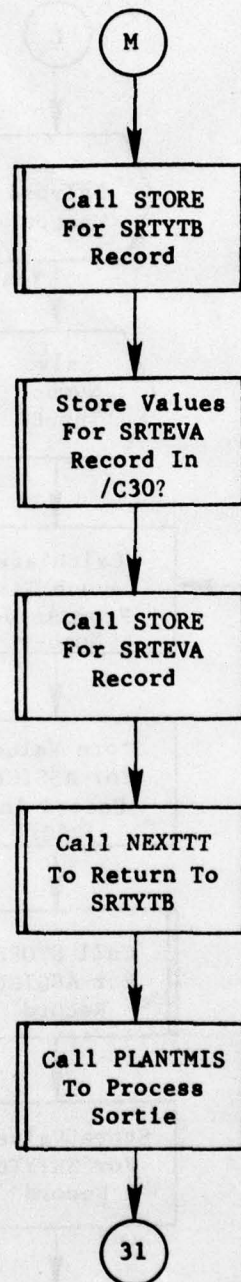


Figure 70. (Part 16 of 30)

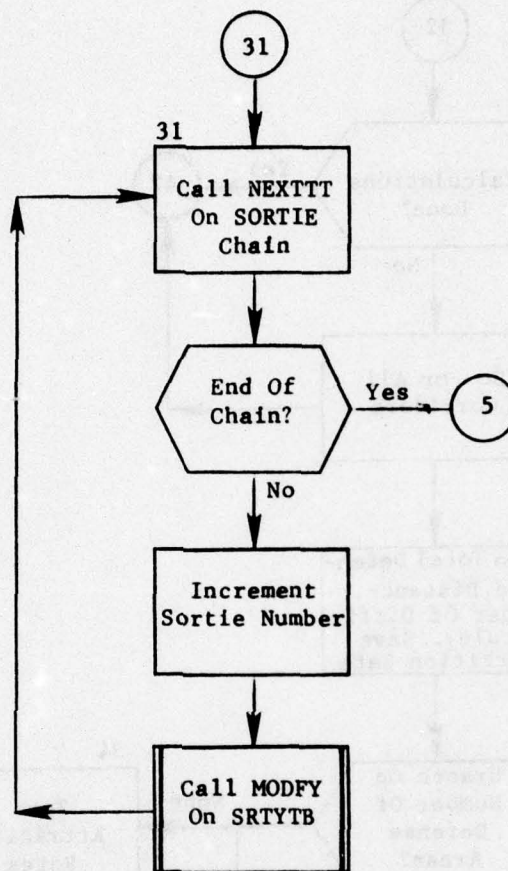


Figure 70. (Part 17 of 30)

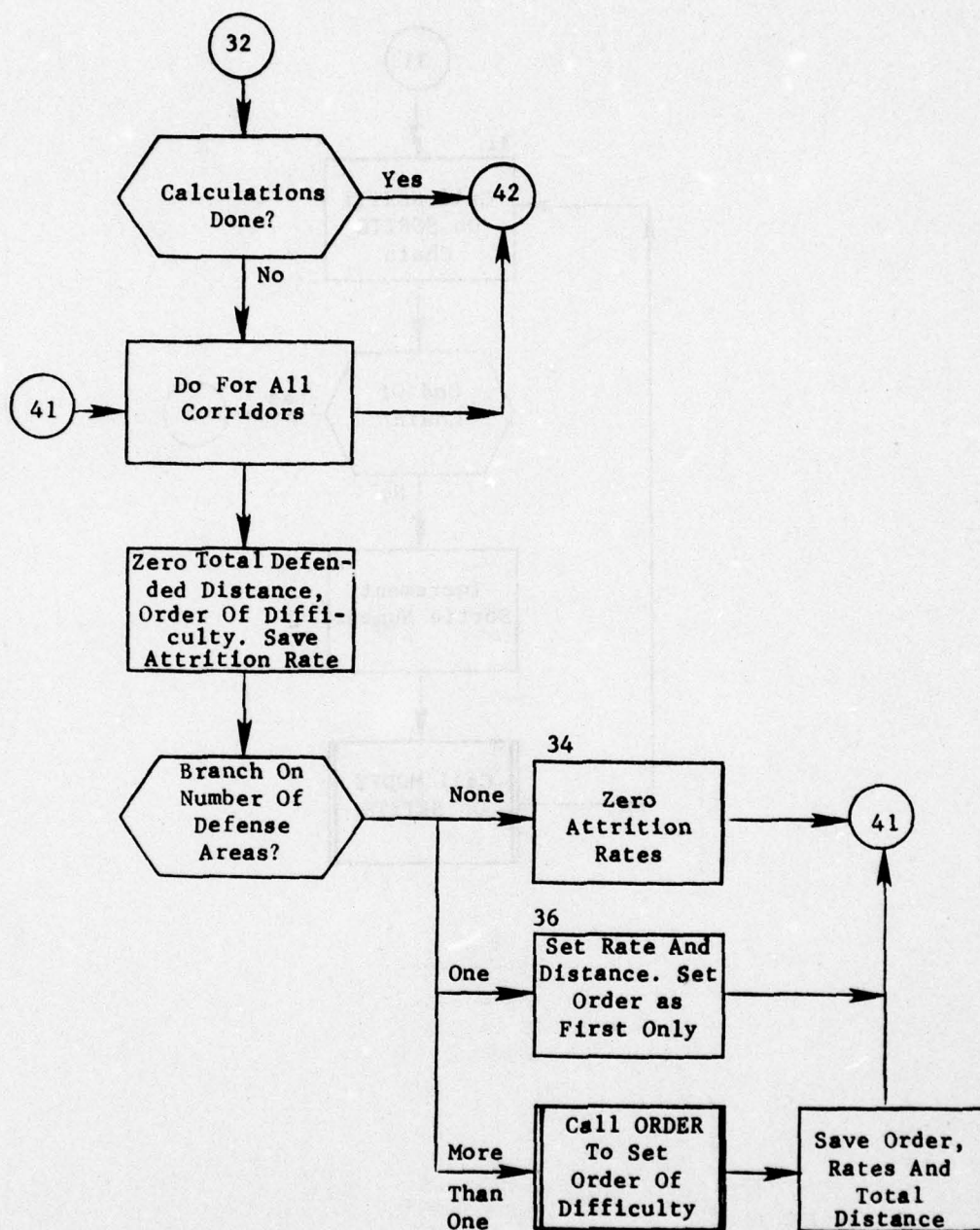


Figure 70. (Part 18 of 30)

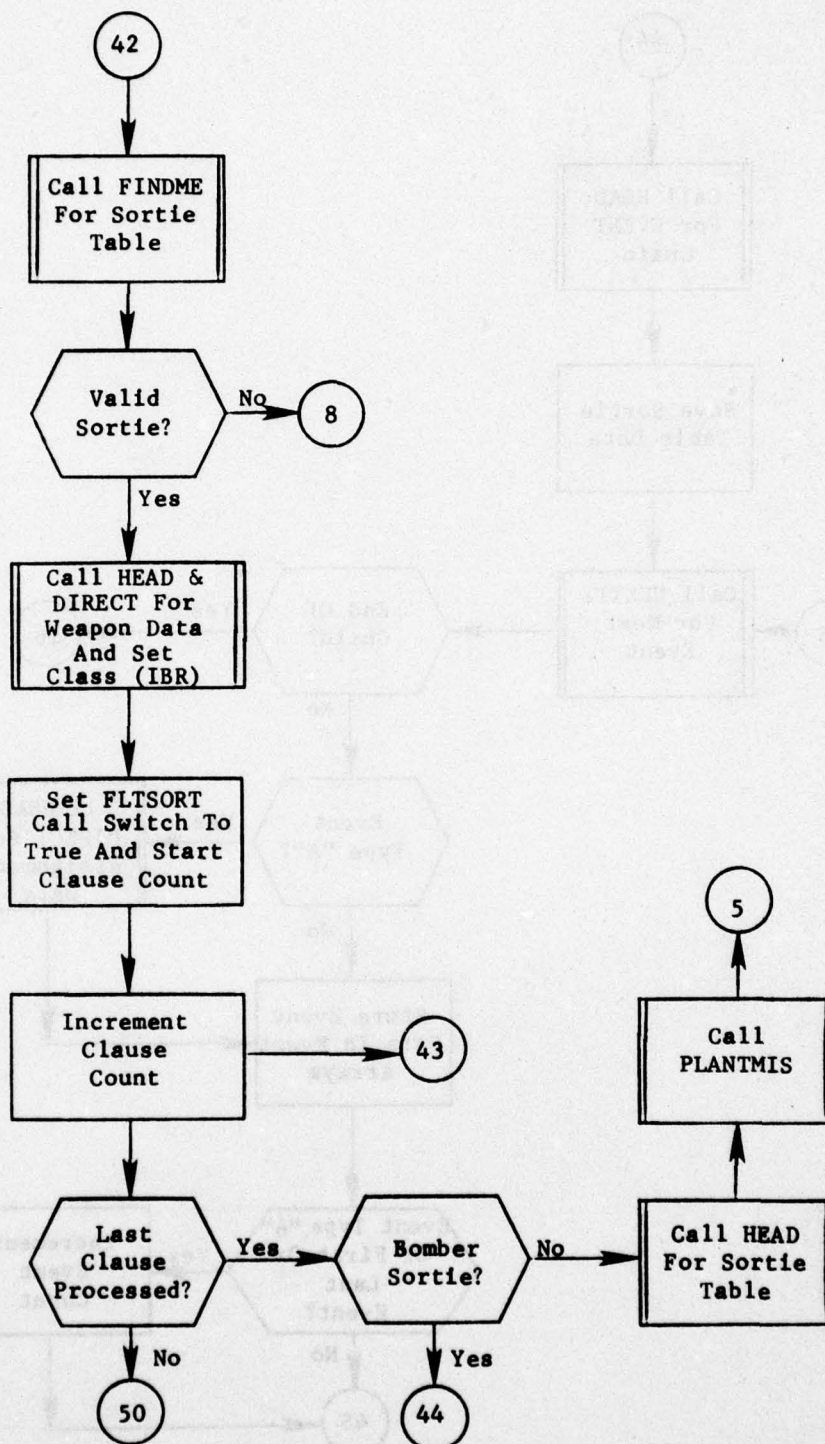


Figure 70. (Part 19 of 30)

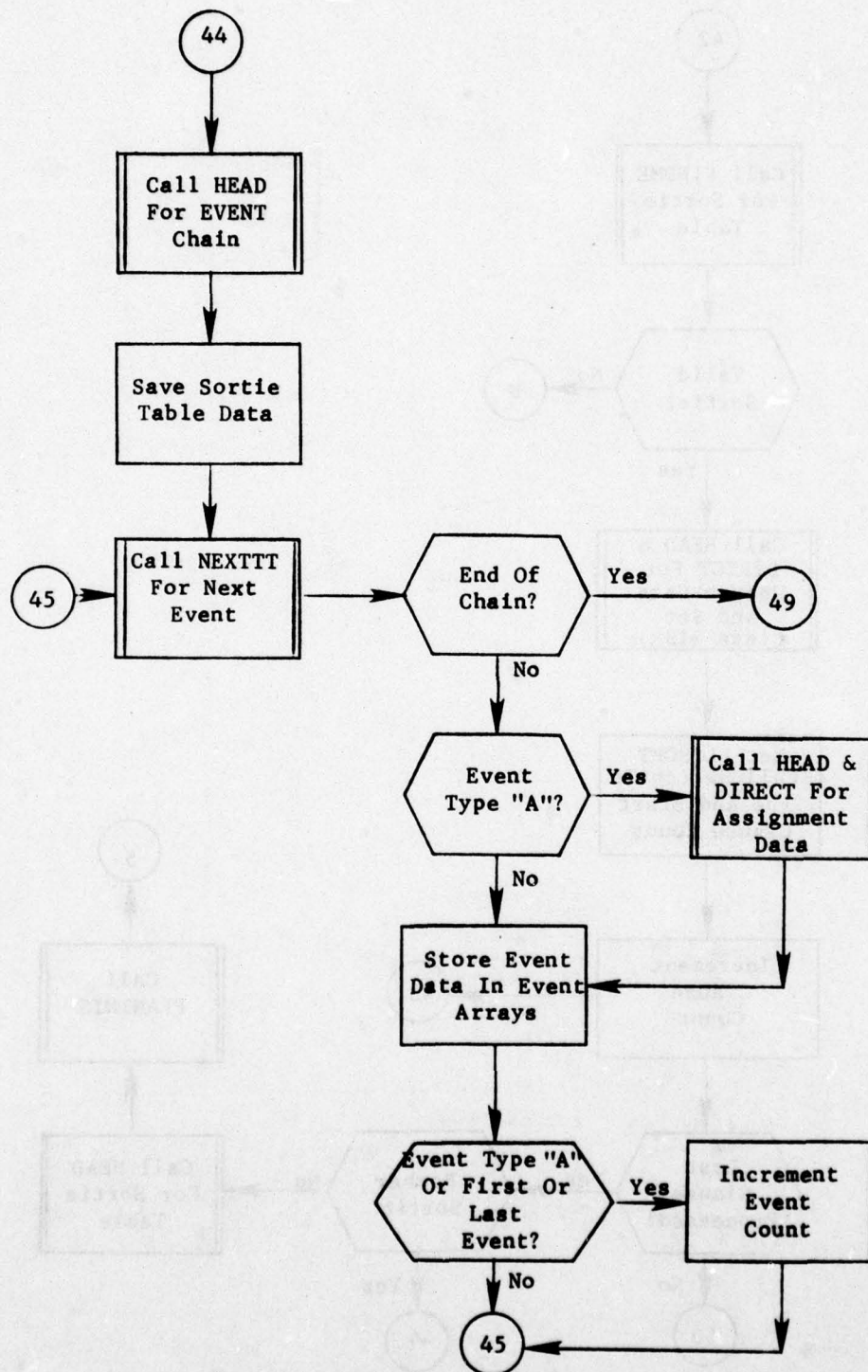


Figure 70. (Part 20 of 30)
366

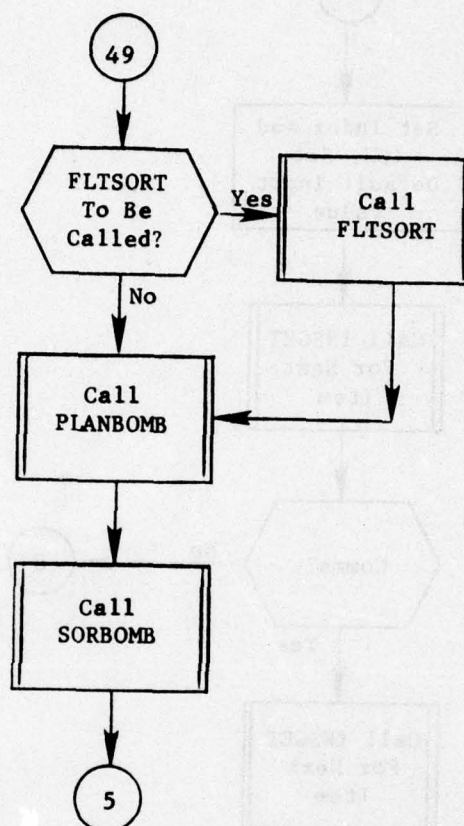


Figure 70. (Part 21 of 30)

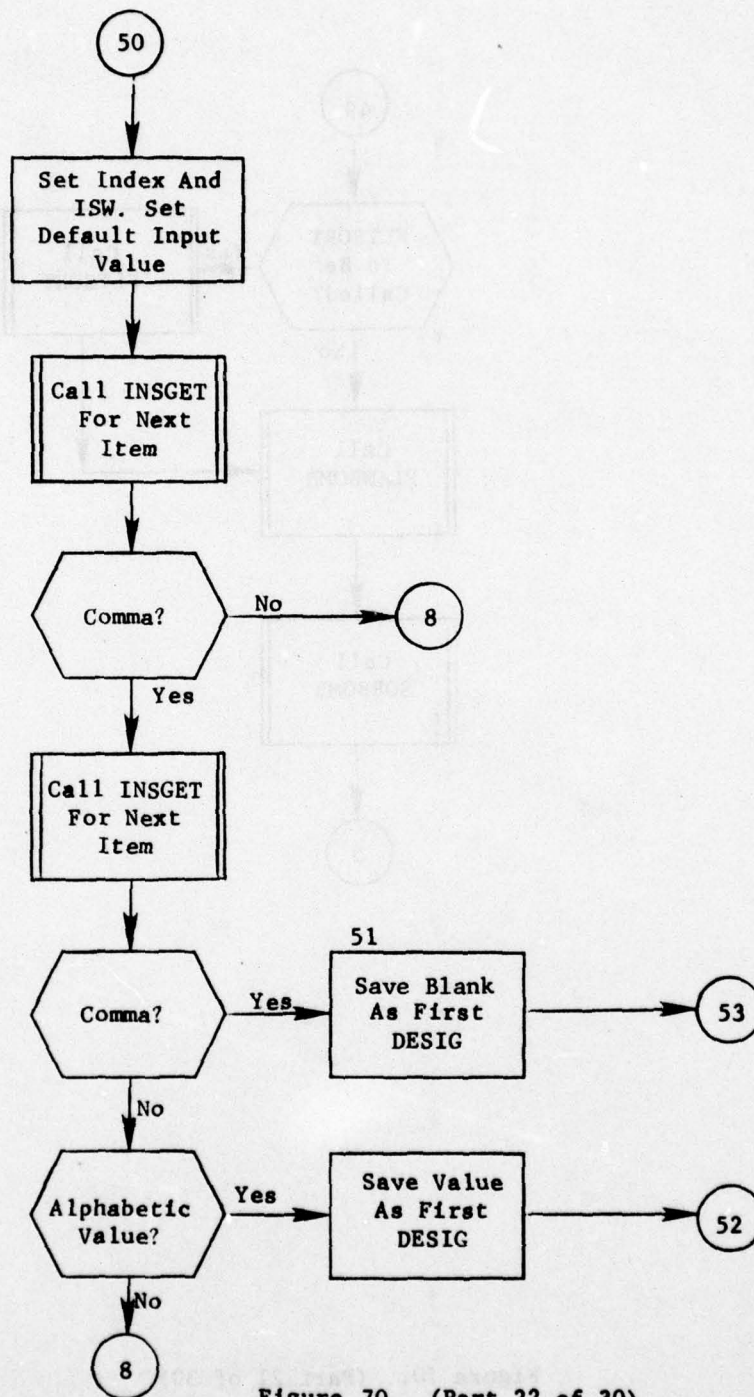


Figure 70. (Part 22 of 30)

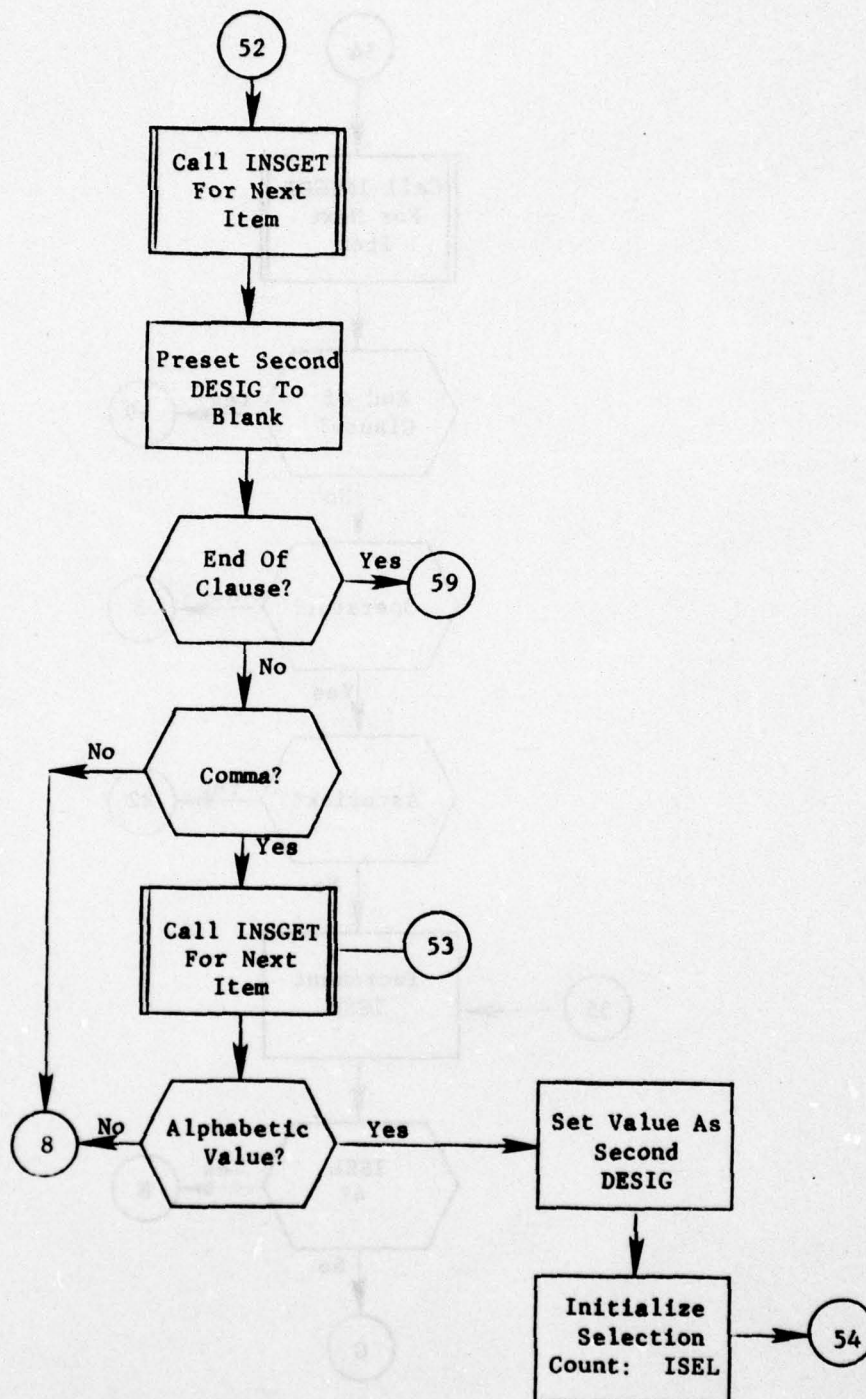


Figure 70. (Part 23 of 30)

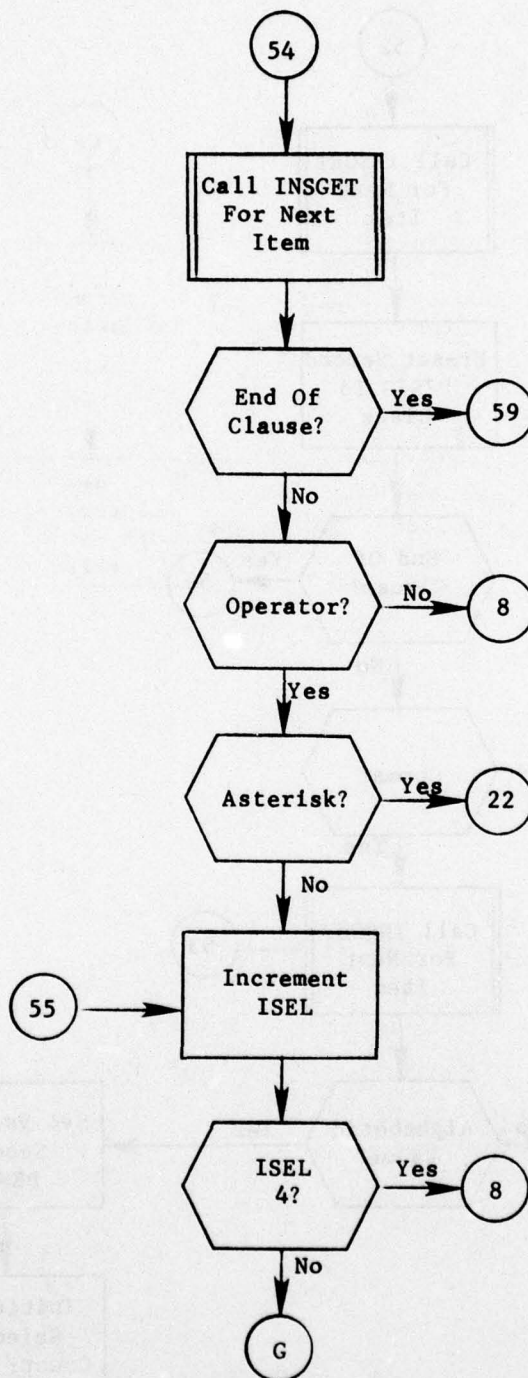


Figure 70. (Part 24 of 30)

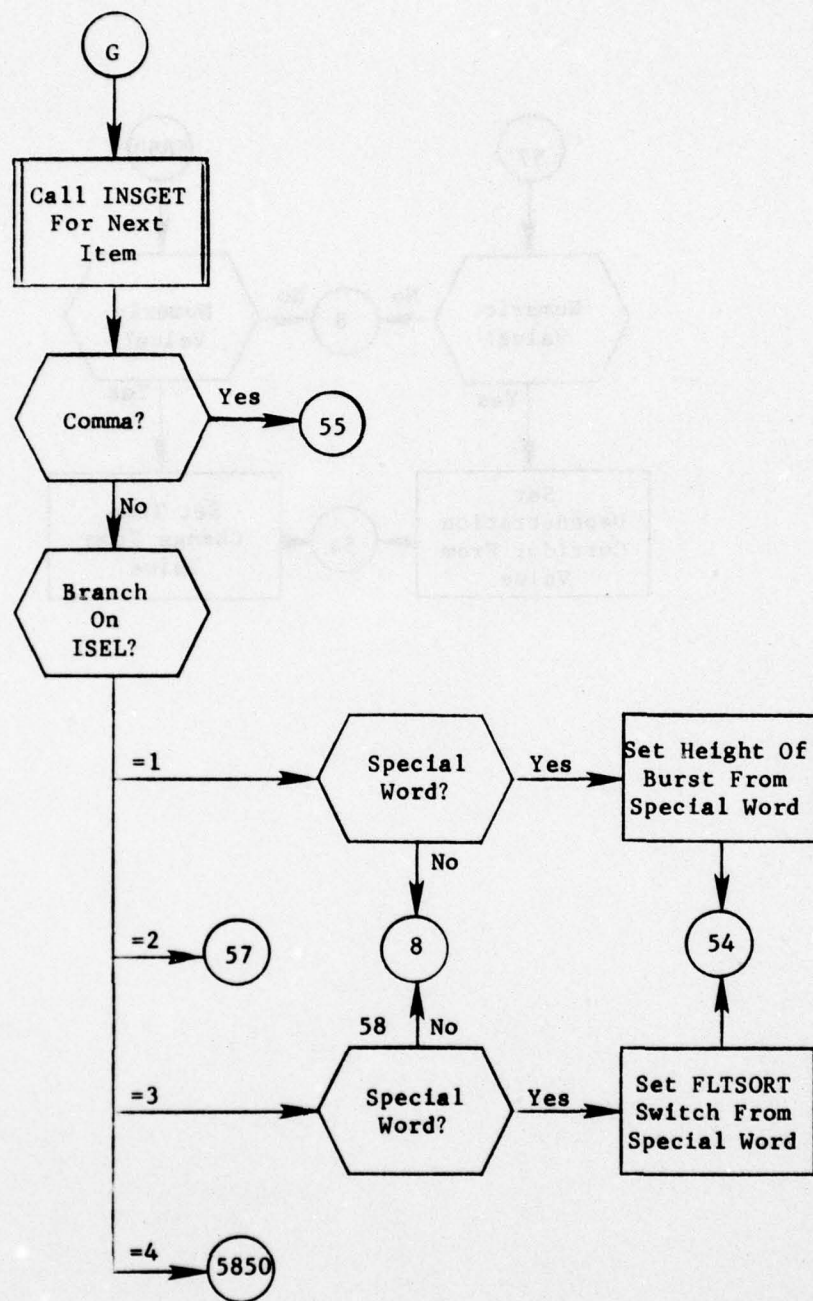


Figure 70. (Part 25 of 30)

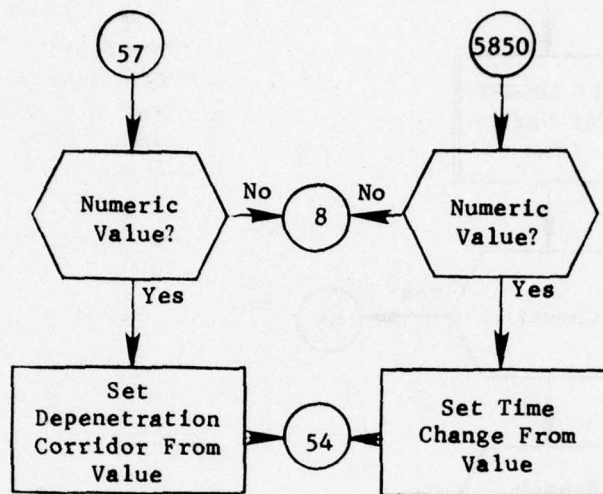


Figure 70. (Part 26 of 30)

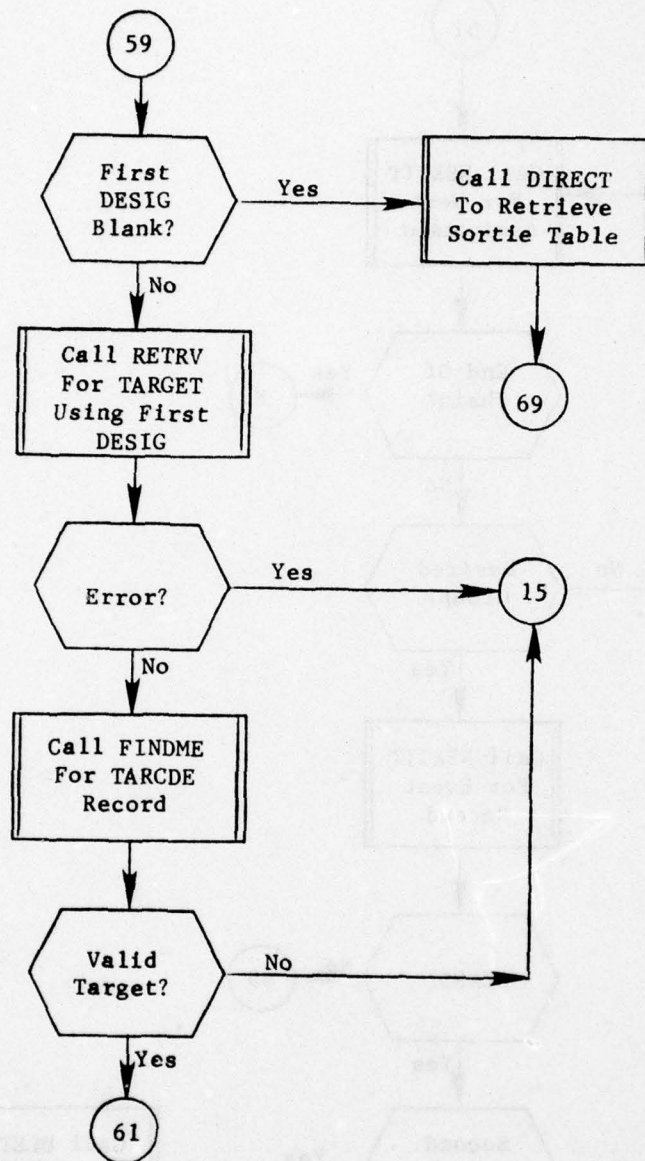


Figure 70. (Part 27 of 30)

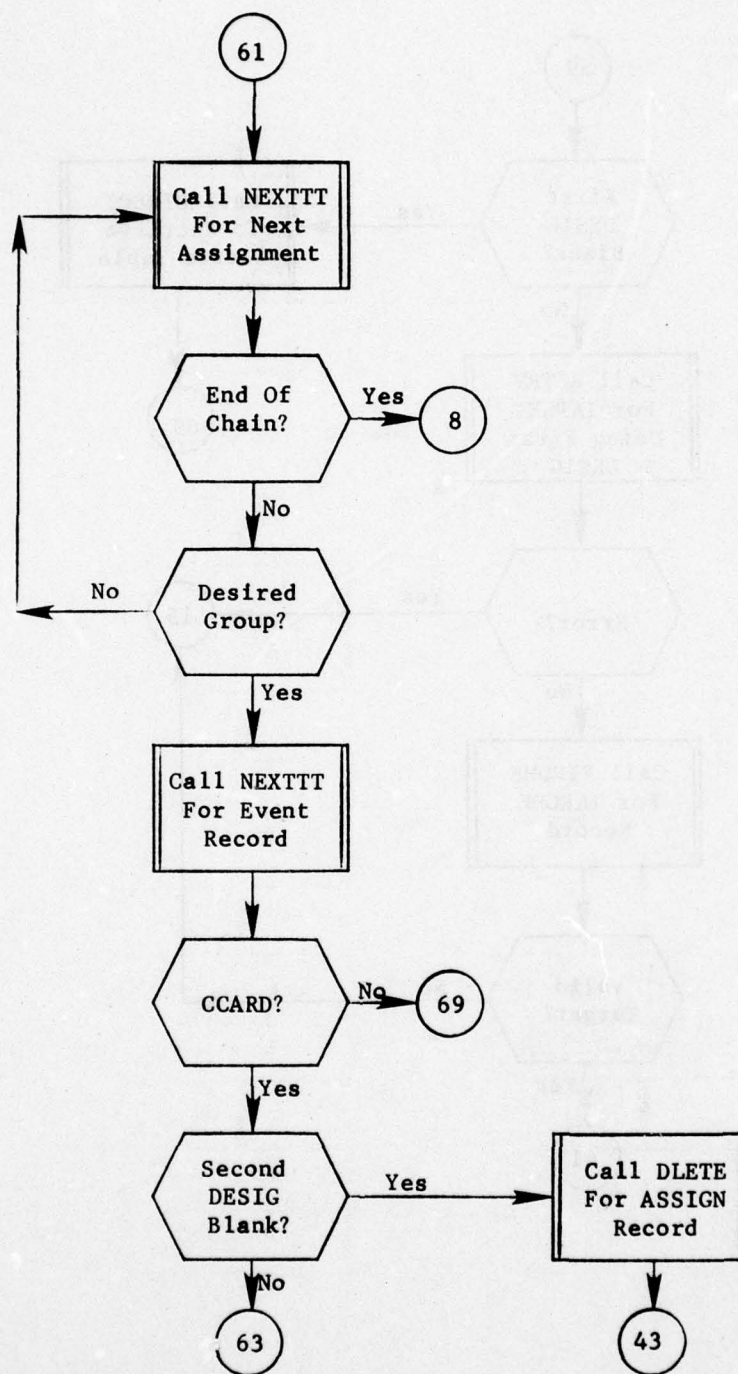


Figure 70. (Part 28 of 30)

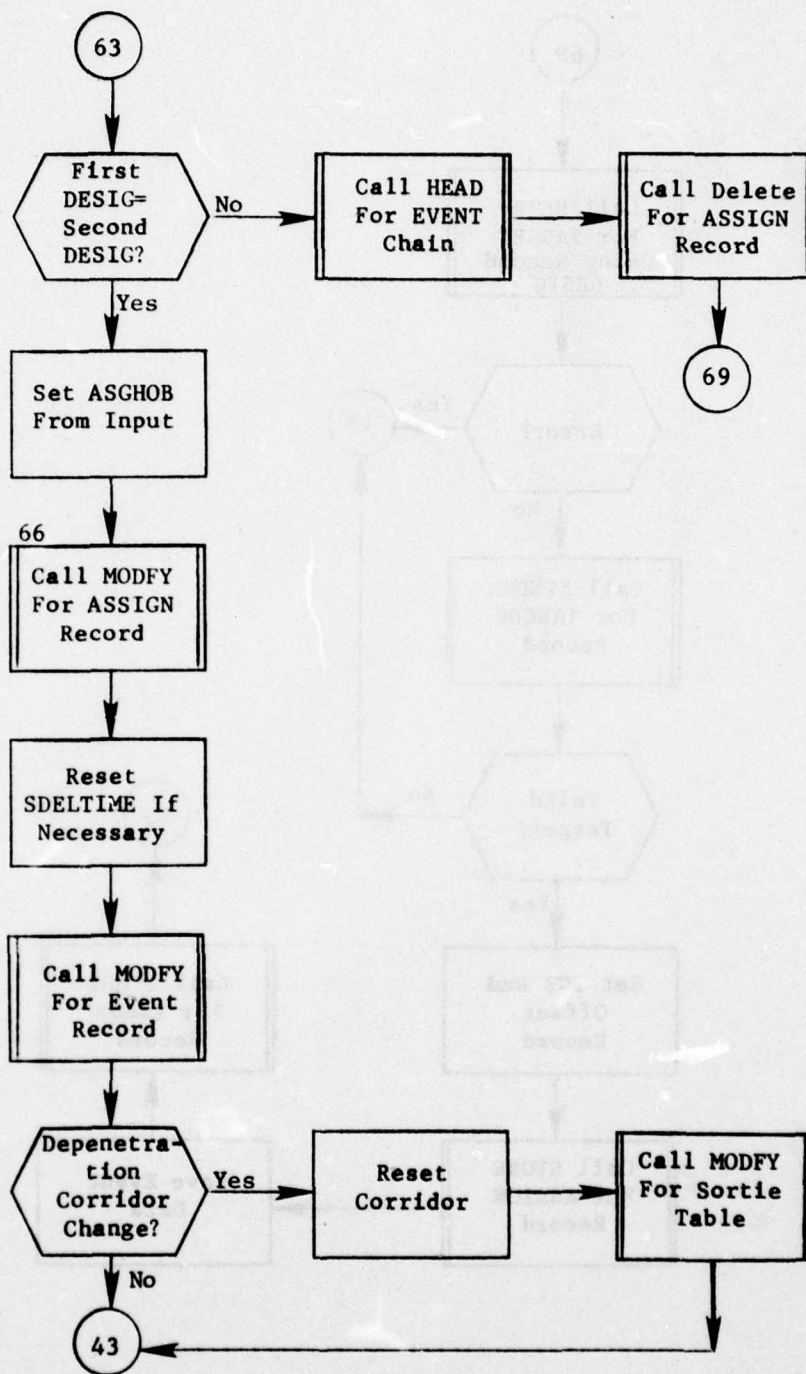


Figure 70. (Part 29 of 30)

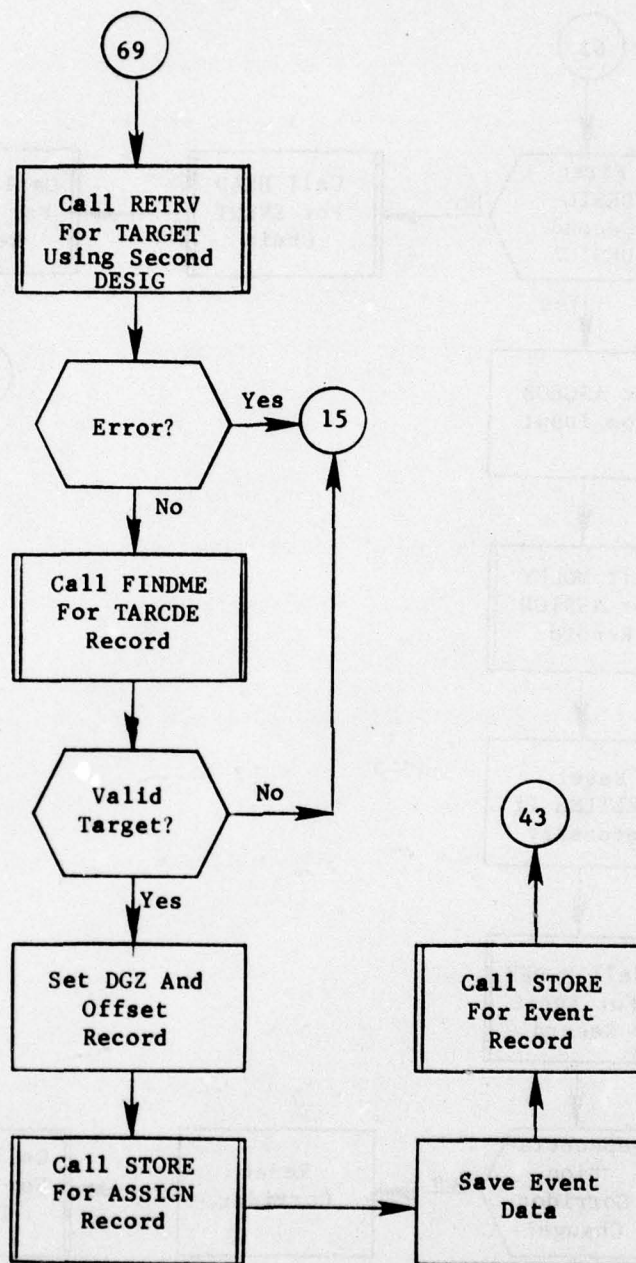


Figure 70. (Part 30 of 30)

4.8.2 Subroutine ADJUST

PURPOSE: To examine the target section of the plan to determine where GO HIGH and GO LOW events are to be placed with respect to the target events, and to adjust these as appropriate.

ENTRY POINTS: ADJUST

FORMAL PARAMETERS: LOWFLAG - Low-altitude flight indicator
PAYALT - Bomber Release altitude indicator

COMMON BLOCKS: ASMARRAY, DINDATA, DISTC, EVENTS, HILO, IGO, OUTSRT, POLITE, VICINITY

SUBROUTINES CALLED: DISTF, INTERP, SNAPIT

CALLED BY: PLAN

Method:

Subroutine ADJUST allocates the GO LOW ranges of G_2 (low-altitude range before the first target) and G_3 (low-altitude range after the first target) beginning at the corridor origin and covering the entire target area. The values for G_1 , G_2 , and G_3 are input from POSTALOC; the G_1 is allocated by blocks 27 and 30 of PLAN. ADJUST is called by PLAN just before the target list is processed. ADJUST begins by calculating the distance from event I to event I+1 in /ASMARRAY/ and storing it in DISTC(1). The initial GO LOW point is then determined from the value of G_2 .^{*} If $G_2 > 0$, the GO LOW will occur G_2 miles before the first target. Here, the first target is defined to mean the first bomb target or the first ASM launch point after the corridor origin. If G_2 is such that the GO LOW point is within 15 minutes (GOMIN) of the

^{*}However, G_2 may be modified if PAYALT is LOW, as described later.

corridor origin, it is extended so that the go low occurs at the origin. If it is to go low at the origin according to G_2 , any go high event posted at the end of PLAN's block 30 is cancelled and the go low event for G_2 is omitted. If an ASM Launch had been scheduled at the origin, and a go low is also to occur there, the ASM Launch point is recalculated to occur 5 minutes after the origin along the original flight path.

For plans in which $G_2 = 0$, the bomber will go low at the first target, provided that the range to be flown at low altitude after the first target (G_3) > 0 . If G_3 also equals 0, it will fly the entire mission at high altitude. If $G_2 < 0$, it will fly $-G_2$ miles beyond the first target before going low. The total low-altitude range in this case is $G_3 - (-G_2)$ miles.

Once the go low point has been found, the number of the event preceding the altitude change is stored in ISTORELO and the distance from that event to the go low is stored in FACLO. The point at which the go high event will occur then is determined by subtracting the distances in array DISTC from the available go low range. When the range becomes negative or zero, the index to the array will be set to the number of the event preceding the altitude change. This number is stored in ISTOREHI and the distance from that event to the go high event is stored in FACHI.

These preliminary locations must now be checked to ensure that the bomber does not change altitude within a critical distance of a target or ASM Launch. These distances are described by the variables VHB, VHA, VLB, and VLA, contained in common /VICINITY/. These variables represent the mileages which correspond to the constant time parameters THB, THA, TLB, and TLA. The settings in PLNTPLAN for these parameters are shown in figures 71 and 72. If a change does occur within a critical distance, the values of ISTORELO (ISTOREHI) and FACLO (FACHI) are adjusted so that the altitude change is moved to a point which is the required distance away from the target.

The distance flown at low altitude never is decreased by the move. For example, if the bomber originally were to go low less than VLA miles after the target, the altitude change is moved to a distance VLB miles before the target. If the critical distances to two (or more) targets overlap, the altitude change is moved, either forward or backward as the situation requires, past the entire cluster of targets.

In making these adjustments the amount of low altitude flight may be increased. This is illustrated by the example shown in figure 73. It shows two targets T_1 and T_2 with their associated neighborhoods drawn taking account of the parameters in figures 71 and 72 and a section of bomber path shown by a dotted line. In this case, a GO HIGH found, say at point p, would be moved first to point q, and finally to point r. The time of low-altitude flight would be increased, in this case, to at most twice the sum of THB + THA. For this to occur, targets would have

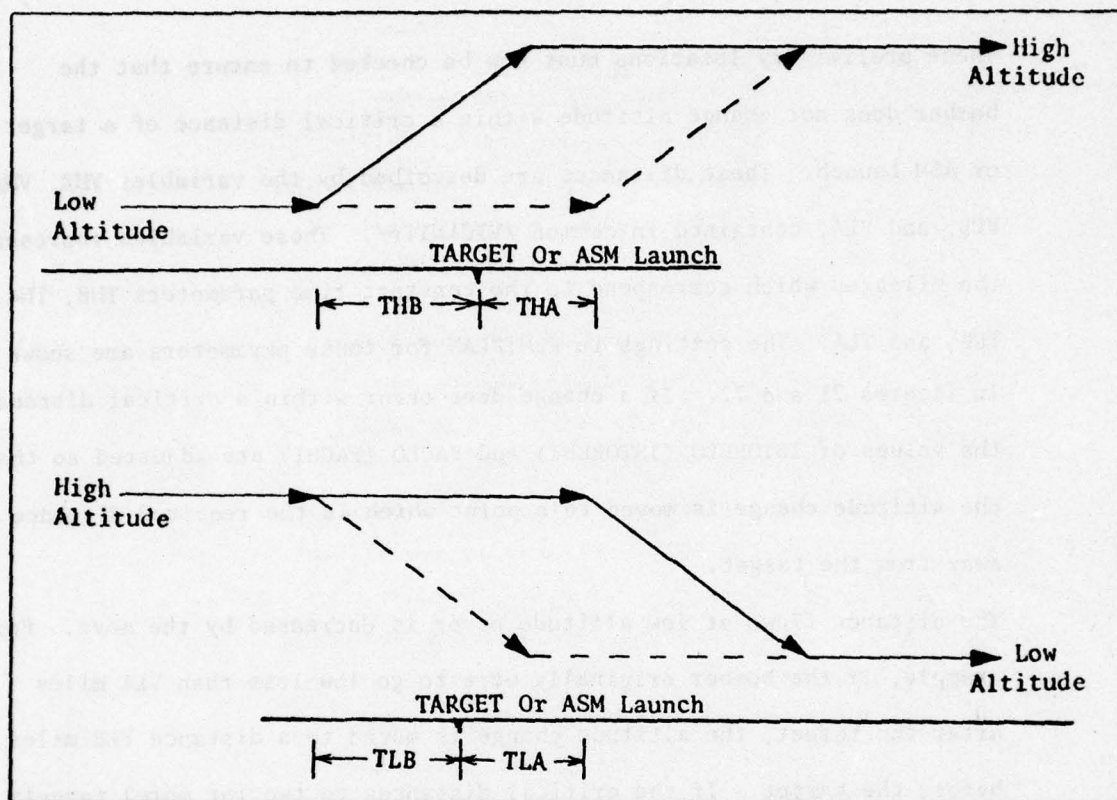


Figure 71. High-Altitude Adjustment

PARAMETER	TIME (MINUTES)	DESCRIPTION
THB	15	Time before a target during which the bomber may not go high
THA	2	Time after a target during which the bomber may not go high
TLB	10	Time before a target during which the bomber may not go low
TLA	3	Time after a target during which the bomber may not go low

(Variables VHB, VHA, VLB, and VLA in common /VICINITY/ represent the mileages which correspond to the time parameter THB, THA, TLB, and TLA)

Figure 72. Low-Altitude Adjustment

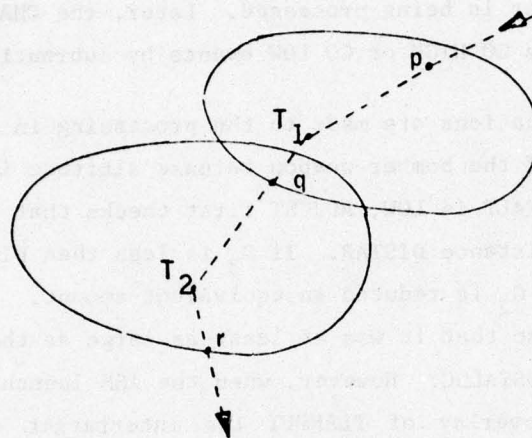


Figure 73. Increase In Low-Altitude Flight

to be within $THB + THA$ minutes of flying time to each other.

Communication between subroutine ADJUST and subroutine PLAN is established through common /HILO/. The final values of ISTOREHI, ISTORELO, FACHI, and FACLO are used by the main program to insert the CHANGALT events as the target list is being processed. Later, the CHANGALT events will be interpreted as GO HIGH or GO LOW events by subroutine SWTCHALT.

Slight modifications are made to the processing in ADJUST, as described previously, if the bomber weapon release altitude indicator PAYALT is not HIVAL. If PAYALT is LOW, ADJUST first checks that G_3 is greater than the intertarget distance DISTAR. If G_3 is less than DISTAR, G_3 is increased to DISTAR and G_2 is reduced an equivalent amount. Module POSTALOC had generated G_3 so that it was at least as large as the intertarget distance computed in POSTALOC. However, when the ASM launch points are recomputed in the PLNT overlay of PLANOUT the intertarget distance (i.e., the interweapon release point distance) may be increased. In this case, ADJUST increases the intertarget low-altitude range allocation (G_3) to cover the required distance. If the sum of G_2 and G_3 is less than DISTAR, then G_3 is set to DISTAR and G_2 is set to zero. After modifying G_2 and G_3 if necessary for PAYALT equal LOW, the processing of the sortie by ADJUST is the same as for PAYALT equal HIVAL. Because ADJUST only increases low-altitude flight to cover the targets, the HIVAL processing will not move the low-altitude flight path so that a weapon release would be at high altitude.

If PAYALT is HIGH, ADJUST does not use the normal processing to determine the GO HIGH point. If G_2 is greater than zero, the bomber goes low and then high just prior to the first target. Therefore, if PAYALT is high, ISTOREHI is set to ISTORELO (which is equal to one). The distance FACHI is set so that the bomber goes high just prior to the first target. If the total low flight in this segment (FACHI-FACLO) is less than the minimum low flight distance GOMIN, this segment is ignored by setting

Figure 74 illustrates subroutine ADJUST.

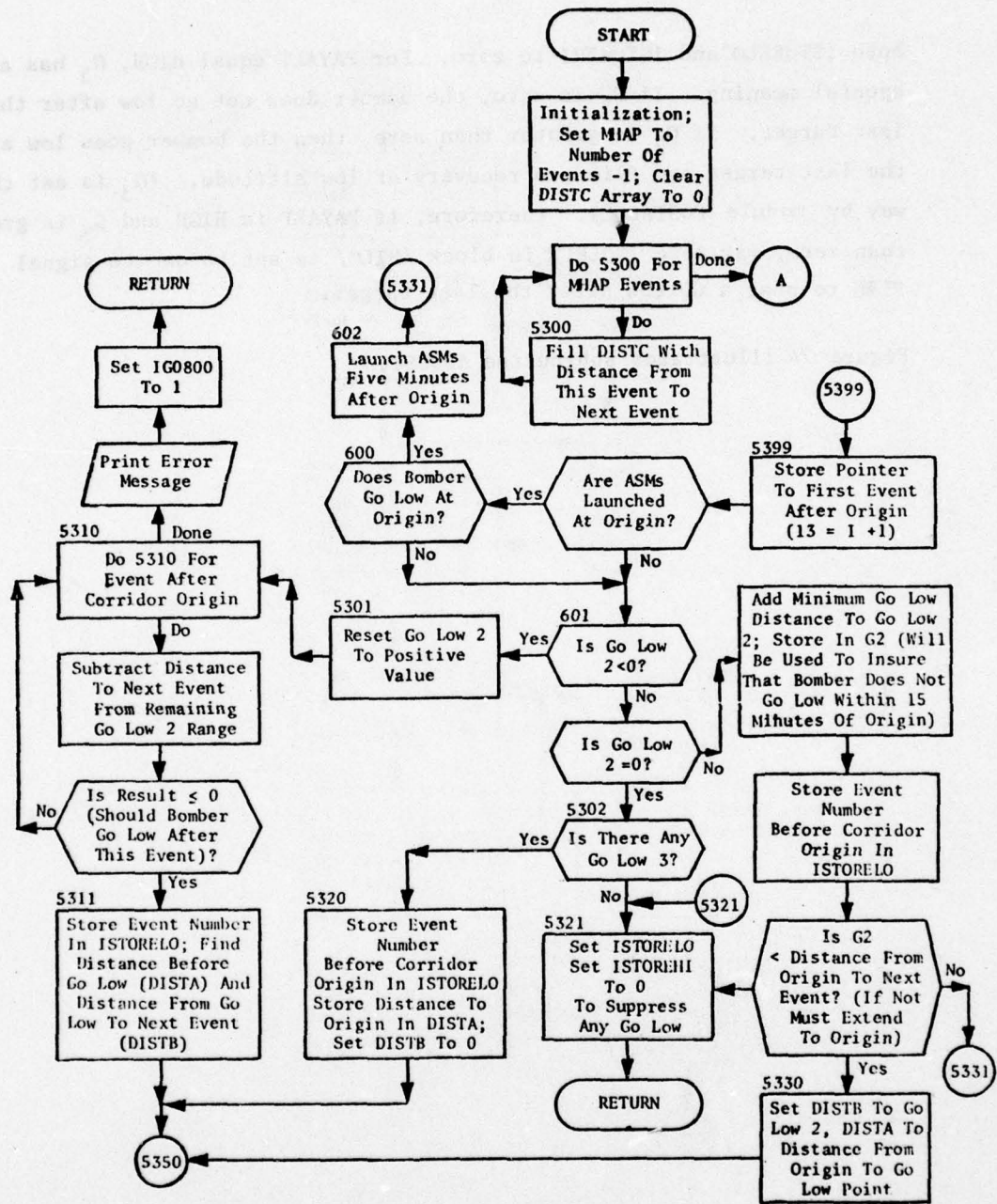


Figure 74. Subroutine ADJUST (Part 1 of 9)

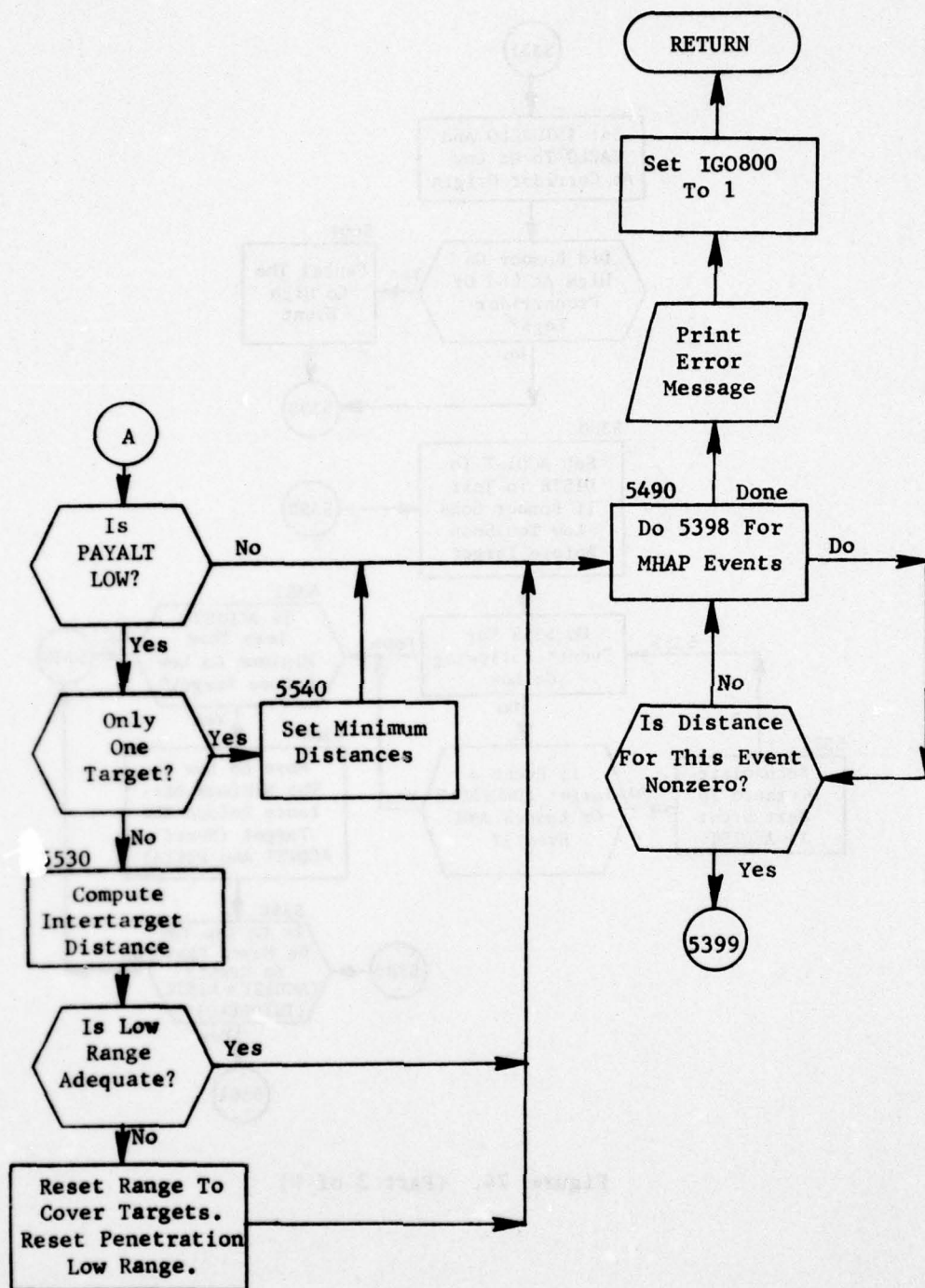


Figure 74. (Part 2 of 9)

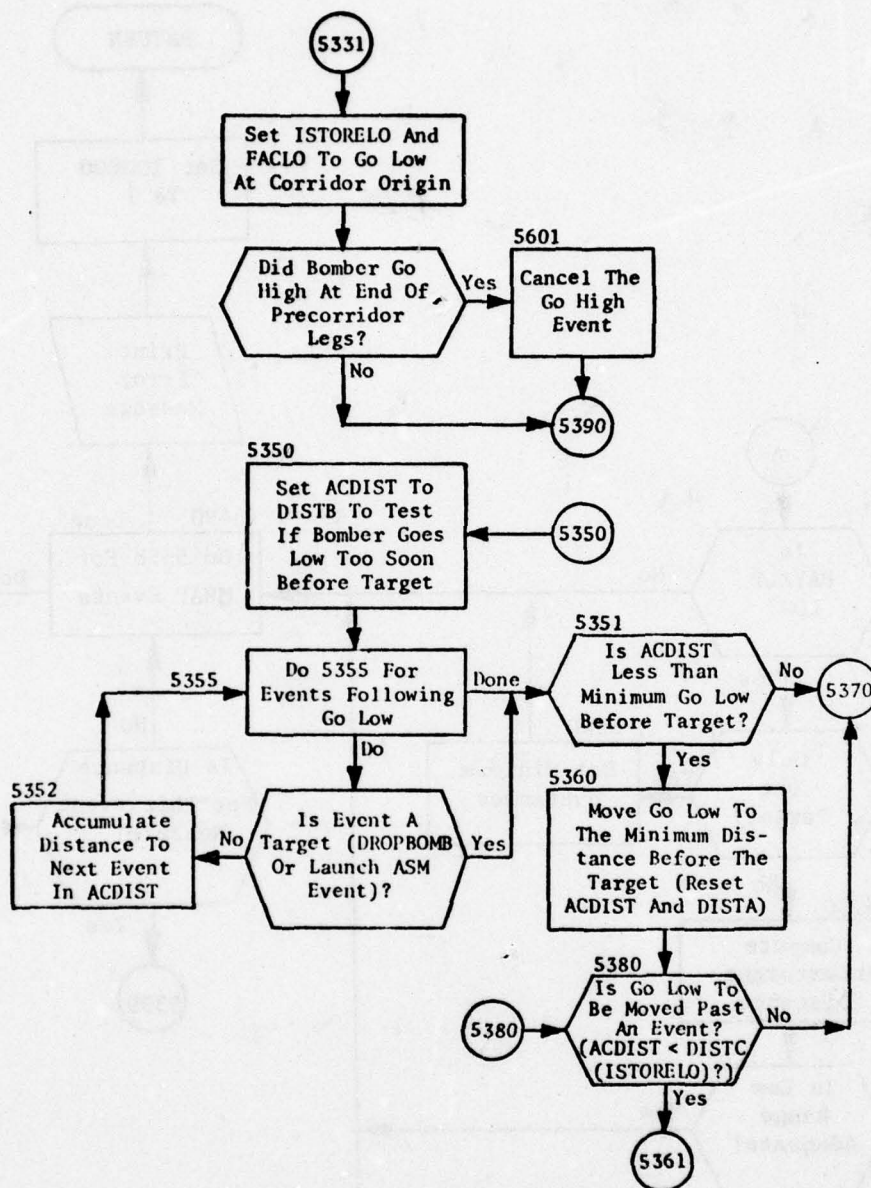


Figure 74. (Part 3 of 9)

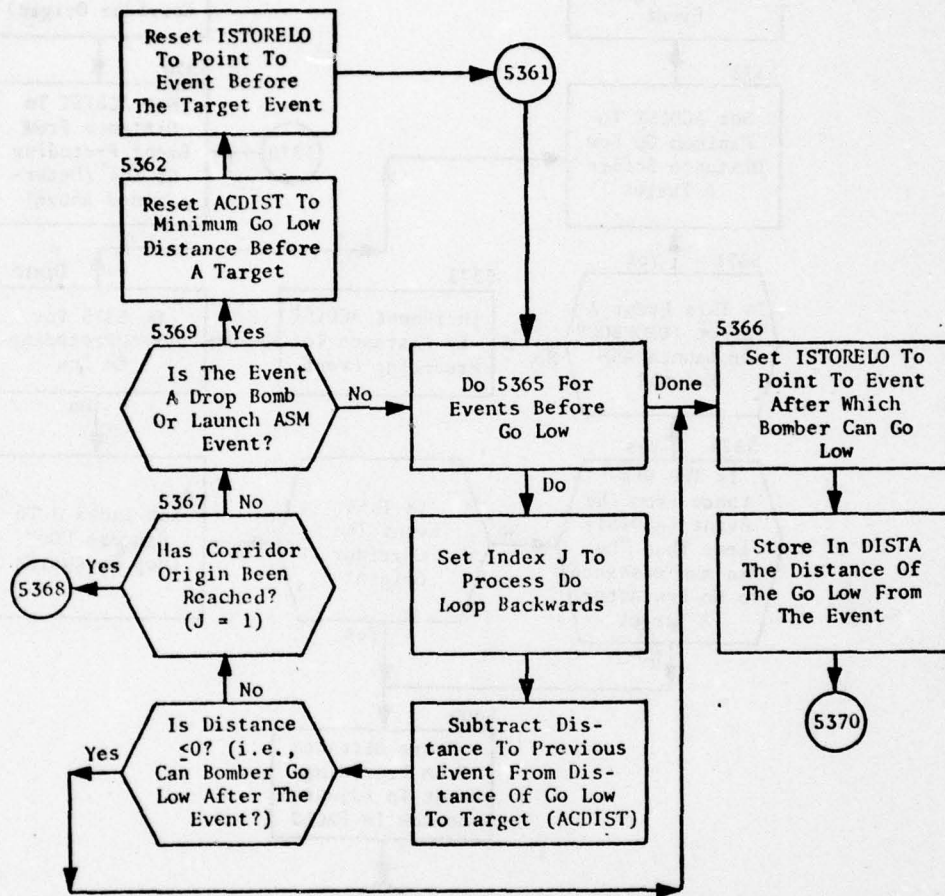


Figure 74. (Part 4 of 9)

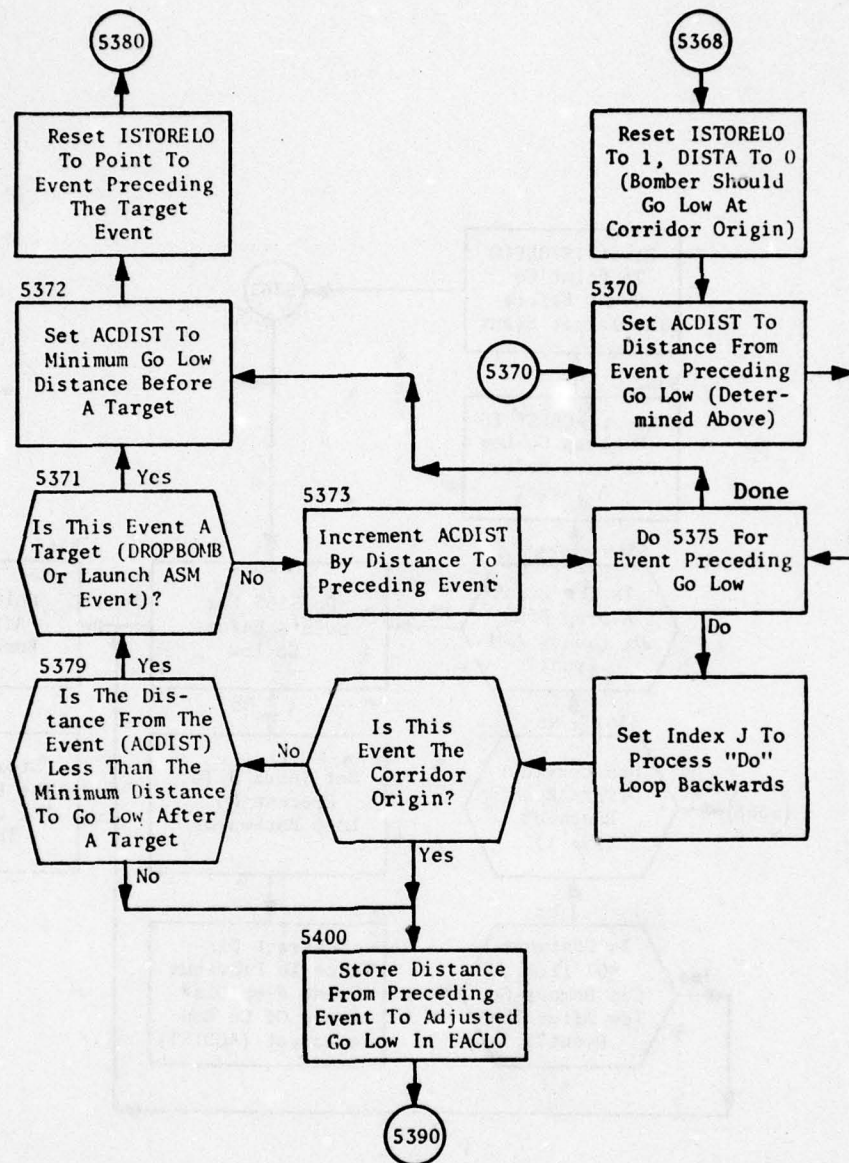


Figure 74. (Part 5 of 9)

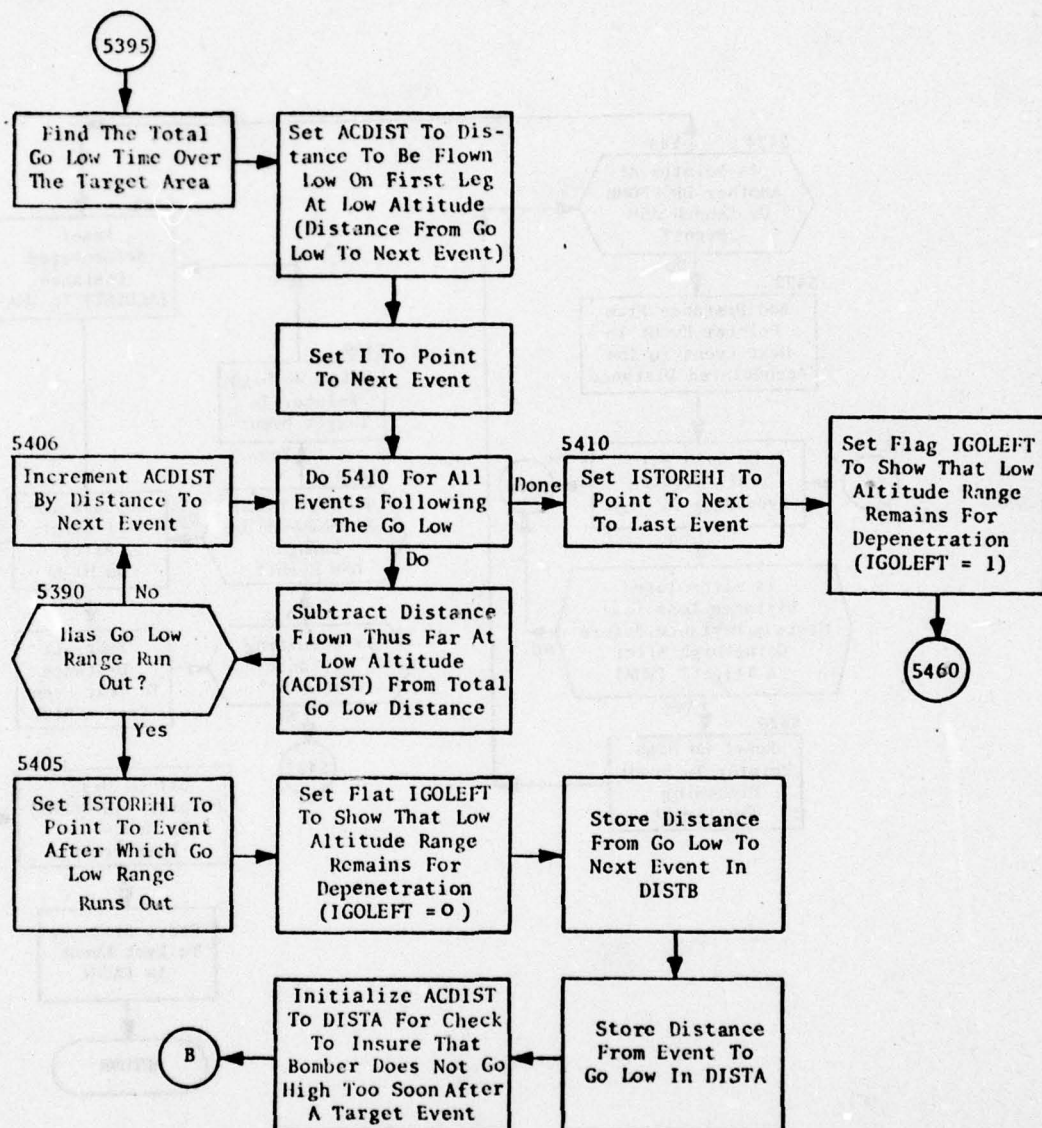


Figure 74. (Part 6 of 9)

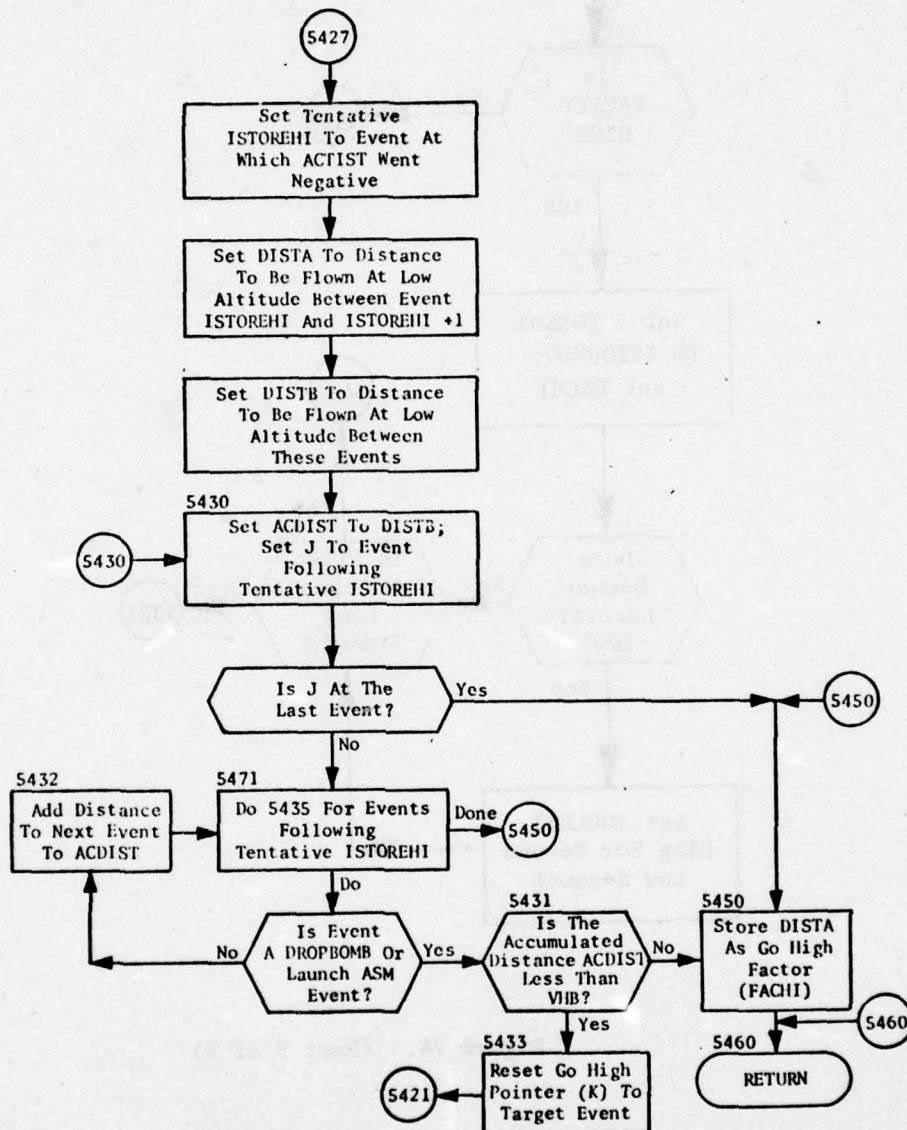


Figure 74. (Part 8 of 9)

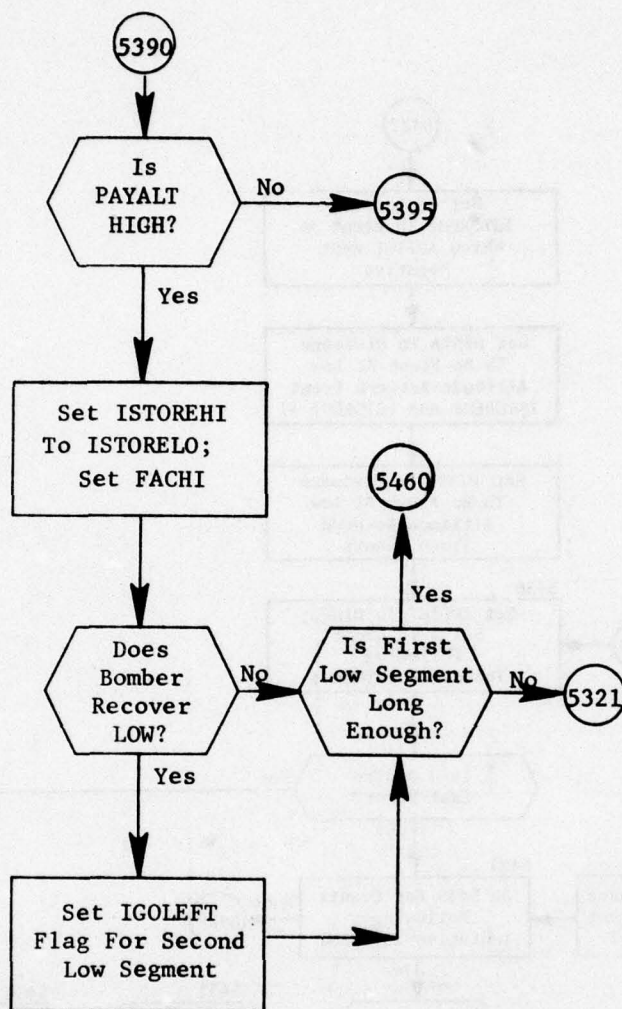


Figure 74. (Part 9 of 9)

4.8.3 Subroutine CHGTIM

PURPOSE: Alter finalized history table by increasing or decreasing times among desired events.

ENTRY POINTS: CHGTIM

FORMAL PARAMETERS: None

COMMON BLOCKS: CONTR1, DINDATA, OUTSRA

SUBROUTINES CALLED: None

CALLED BY: DISTIME

Method:

After subroutine DISTIME converts distances into time increments, CHGTIM is called to process all time changes specified on sortie change cards for this plan. Sortie cards permit time alterations to any DROPBOMB or AIM ASM weapon event. Interceding events will have these times adjusted by an amount proportional to their weight in the sum of the time between events as calculated by DISTIME. In other words, the requested time change for a given weapon event is prorated starting with the last DROPBOMB or AIM ASM event.

Array CHTIM contains time changes for all weapon events (DROPBOMB or AIM ASM). Array HDT contains time increments for all events in the sortie. These events consist of the weapon events plus additional events such as LAUNCH, REFUEL, DOGLEG, GO LOW, GO HIGH, and RECOVER. Also there is an ASM TGT event that defines an ASM flight time. The two arrays are similar in that the first weapon event in array CHTIM and the first weapon event in array HDT are identical as are the second, third, etc. CHGTIM prorates time changes among weapon events. The time increment for event ASM TGT is never changed since this is the ASM flight time; only launch times are changed.

Figure 75 illustrates CHGTIM.

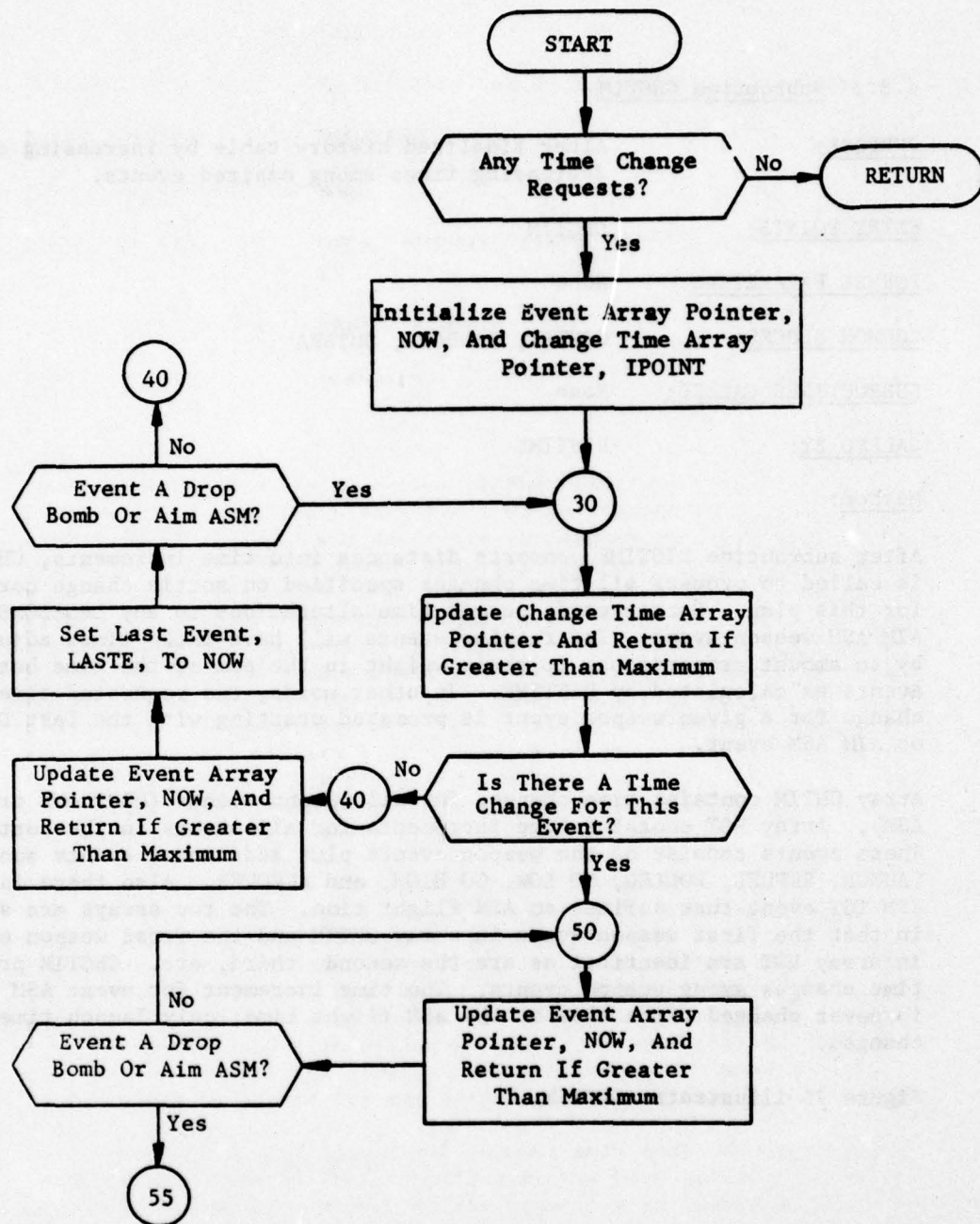


Figure 75. Subroutine CHGTIM
(Part 1 of 2)

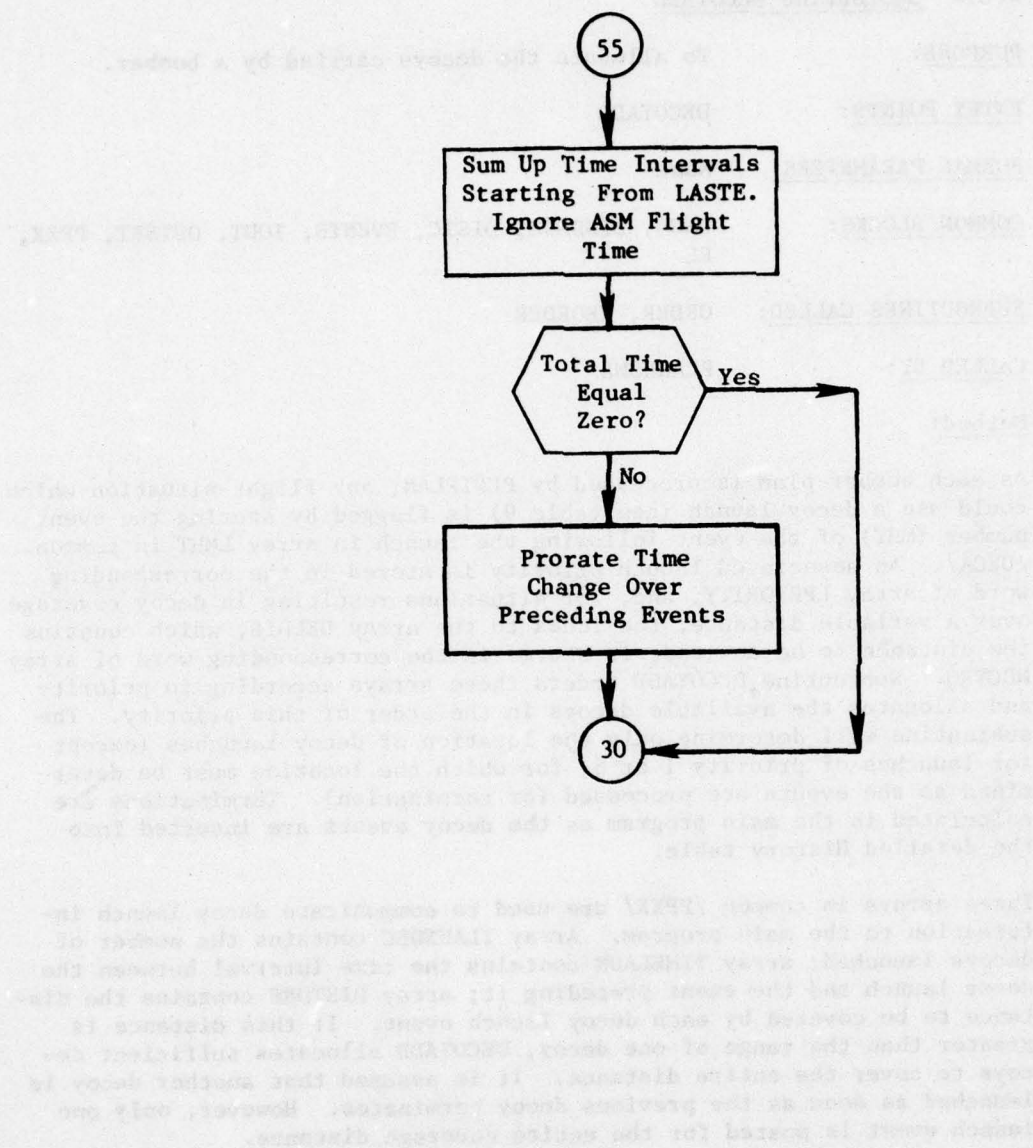


Figure 75. (Part 2 of 2)

4.8.4 Subroutine DECOYADD

PURPOSE: To allocate the decoys carried by a bomber.

ENTRY POINTS: DECOYADD

FORMAL PARAMETERS: None

COMMON BLOCKS: DECA, DINDATA, DISTC, EVENTS, IOUT, OUTSRT, PPXX, RL

SUBROUTINES CALLED: ORDER, REORDER

CALLED BY: PLANBOMB

Method:

As each bomber plan is processed by PLNTPLAN, any flight situation which could use a decoy launch (see table 9) is flagged by storing the event number (MHT) of the event following the launch in array LMHT in common /DECA/. An associated launch priority is stored in the corresponding word of array LPRIORITY, and, for situations resulting in decoy coverage over a variable distance, the index to the array DELDIS, which contains the distance to be covered, is stored in the corresponding word of array NDCYRQ. Subroutine DECOYADD orders these arrays according to priority and allocates the available decoys in the order of this priority. The subroutine will determine only the location of decoy launches (except for launches of priority 1 or 6, for which the location must be determined as the events are processed for termination). Terminations are calculated in the main program as the decoy events are inserted into the detailed History table.

Three arrays in common /PPXX/ are used to communicate decoy launch information to the main program. Array ILAUNDEC contains the number of decoys launched; array TIMELAUN contains the time interval between the decoy launch and the event preceding it; array DISTORE contains the distance to be covered by each decoy launch event. If this distance is greater than the range of one decoy, DECOYADD allocates sufficient decoys to cover the entire distance. It is assumed that another decoy is launched as soon as the previous decoy terminates. However, only one launch event is posted for the entire coverage distance.

Since the bomber must launch all decoys, more than one decoy may be launched at a time if the priority list has been satisfied before all decoys have been allocated. In the case of area coverage, there may not be sufficient decoys remaining to cover the distance of the first allocation. Hence, an entry is made in array DISTORE each time a decoy launch event occurs over the same area. If the last decoy does not cover the same distance as the previous decoy(s), two decoy termination events must be posted for the one launch event.

Table 9. Launch Priority

LAUNCH PRIORITY

CIRCUMSTANCE OF LAUNCH

1

R_L^* miles before first low-altitude gravity bomb attack on a SAM-defended target

2

Immediately before changing from high to low altitude

3

Immediately before penetrating defended airspace if flying at high altitude

4

R_H^{**} miles before first high altitude gravity bomb attack on a SAM-defended target

5

Coverage when flying at high altitude over defended airspace before priority 4 launch

6

R_L miles before subsequent low-altitude gravity bomb attacks on SAM-defended targets

7-8***

Coverage when flying at high altitude over defended airspace after priority 4 launch

* R_L = range of decoy at low altitude (data set to 200 nautical miles)

** R_H = range of decoy at high altitude (data set to 400 nautical miles)

***Priority 8 is used if the coverage is to begin at the point where the priority 4 decoy terminates. Priority 7 is used if the bomber has changed altitude between the priority 4 and the priority 7 launch.

Each time a decoy is allocated, the index to the detailed History table (MHT) is incremented to reserve a line for each event generated by the launch. Since a decoy launched at low altitude (priorities 1 and 6) will always terminate at its target, no termination event is necessary. Hence space is reserved only for the launch event. (This situation is communicated to the termination section by storing the number of decoys launched as a negative number.) For high-altitude launches, either one or two termination events are required in addition to the launch event.

The decoys are allocated by processing each entry in the priority array in order. Since the calculation of timing and distance information differs according to the launch situation, branches are made to various sections of the program according to priority. It should be noted that since the priority 3 launch information is sent to the subroutine as the first instance of a priority 5 launch, the priority 4 launch will be encountered before the priority 3 decoy has, in fact, been allocated. Thus, before the section for priority 4 is processed, a check is made to insure that more than one decoy remains to be allocated. If only one remains, the priority 4 section is skipped, reserving that decoy for the first priority 5 launch (i.e., priority 3).

Since the priority 8 situation calls for the decoy(s) to be launched immediately after the priority 4 decoys to cover the high altitude flight until a go low or a depenetration, the launch event is omitted from the detailed History table and the distance to be covered by the decoys is added onto the distance to be covered by the priority 4 decoys. This merely moves the priority 4 termination event(s) to include the distance that would be covered by the priority 8 decoy(s).

If decoys remain after every entry in the priority array has been processed for the first time, the array will be reprocessed, in order, to provide double coverage. Since many of the values calculated on the first pass need not be recalculated, a different set of branches is taken, according, again, to priority. Up to six allocation passes will be made, as long as decoys remain. If more than six are required to allocate all the decoys, the error message

NUMBER OF DECOY LAUNCHES EXCEEDS CAPACITY OF DECOY
ALLOCATION

is generated. Whenever this occurs, or whenever no more decoys remain to be allocated, control returns to the main program. Subroutine DECOYADD is illustrated by figure 76.

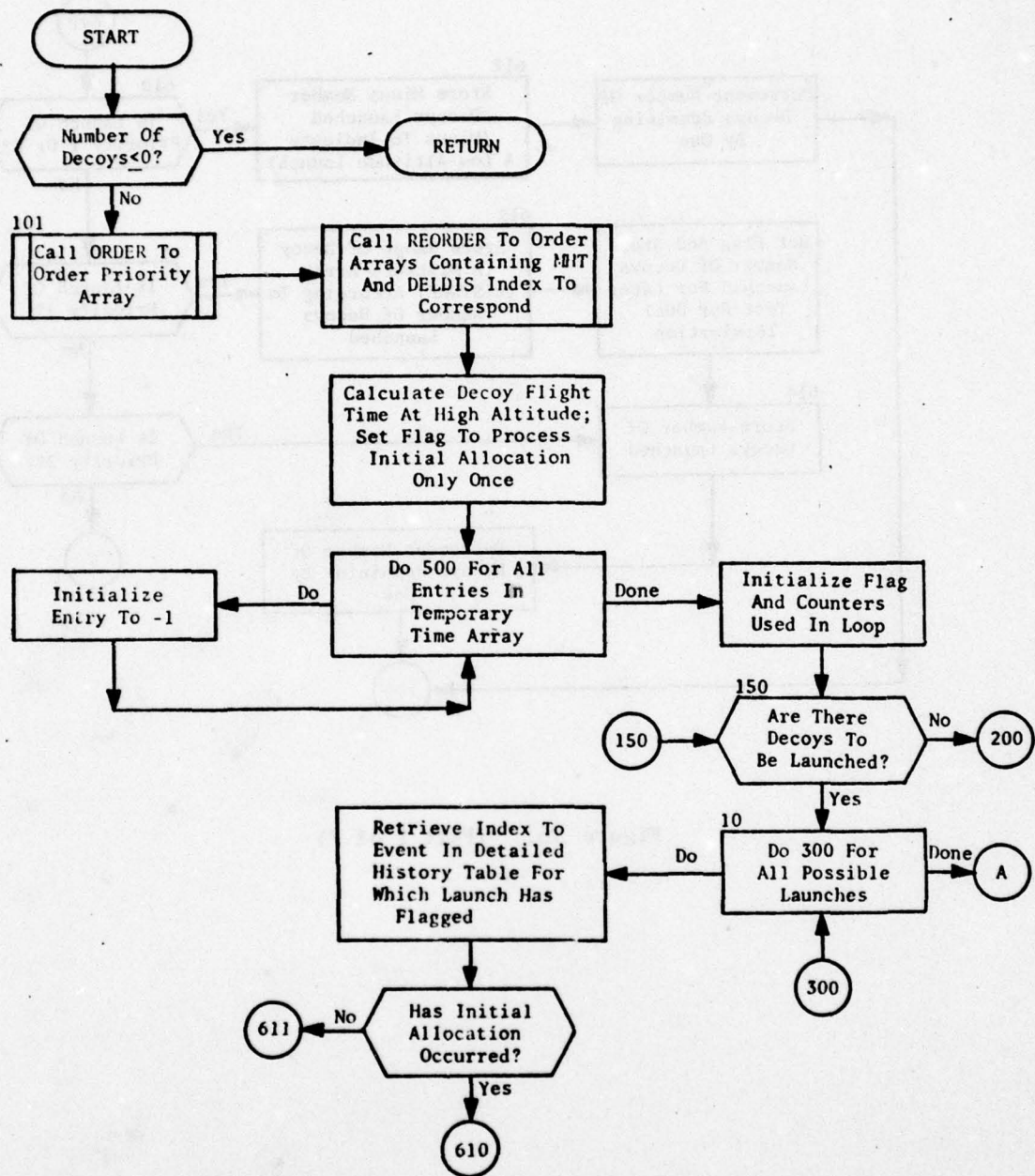


Figure 76. Subroutine DECOYADD
(Part 1 of 7)

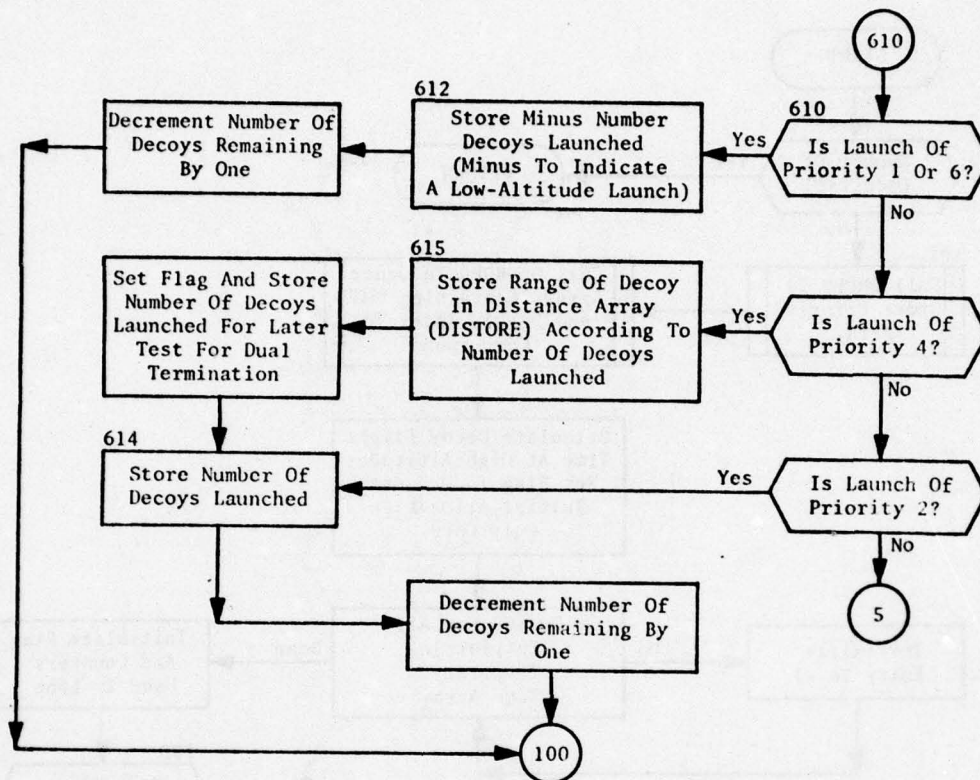


Figure 76. (Part 2 of 7)

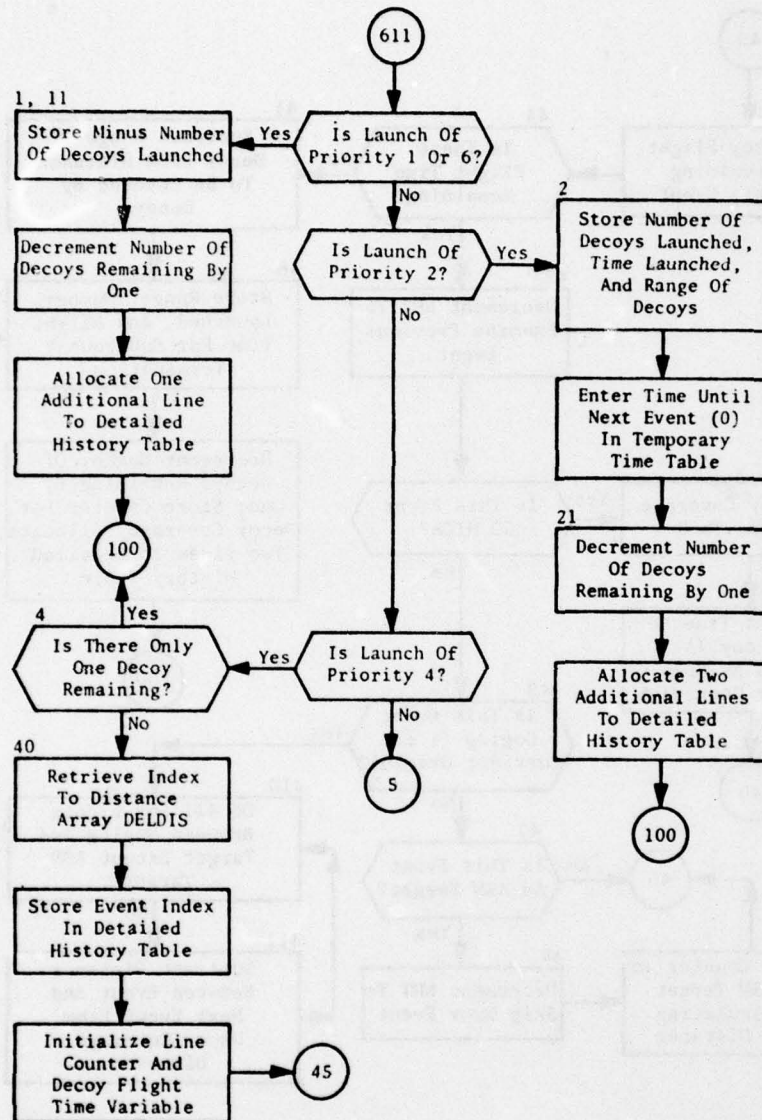


Figure 76. (Part 3 of 7)

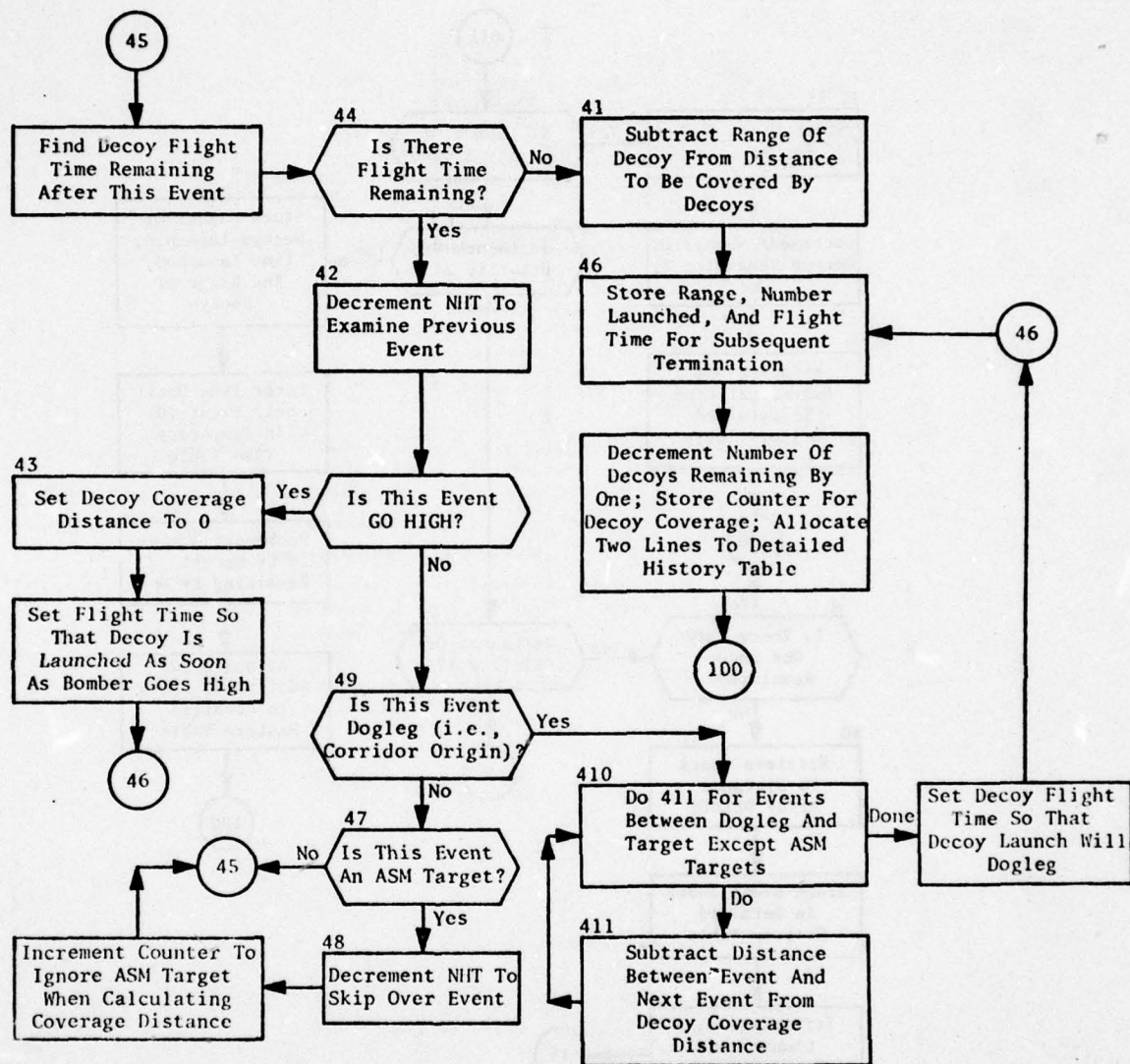


Figure 76. (Part 4 of 7)

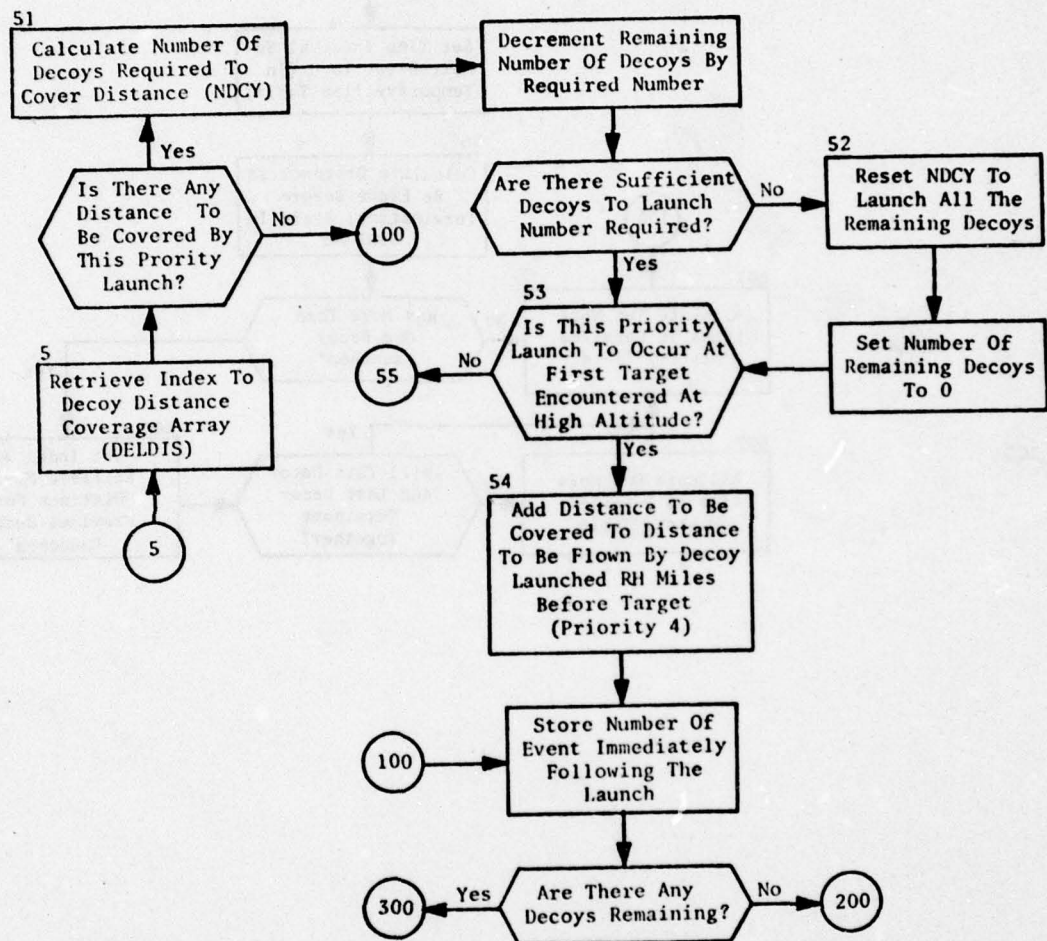


Figure 76. (Part 5 of 7)

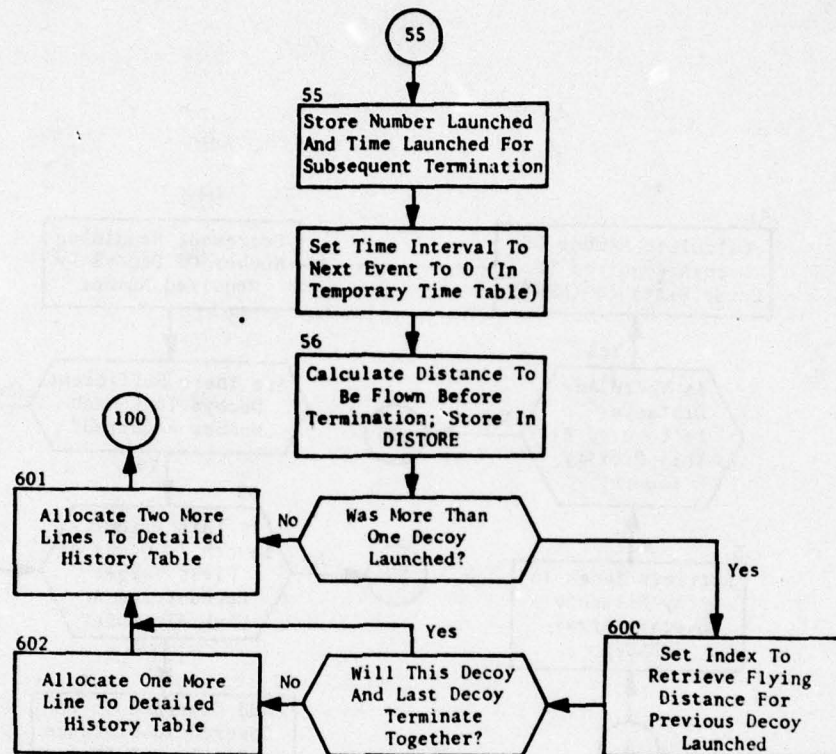


Figure 76. (Part 6 of 7)

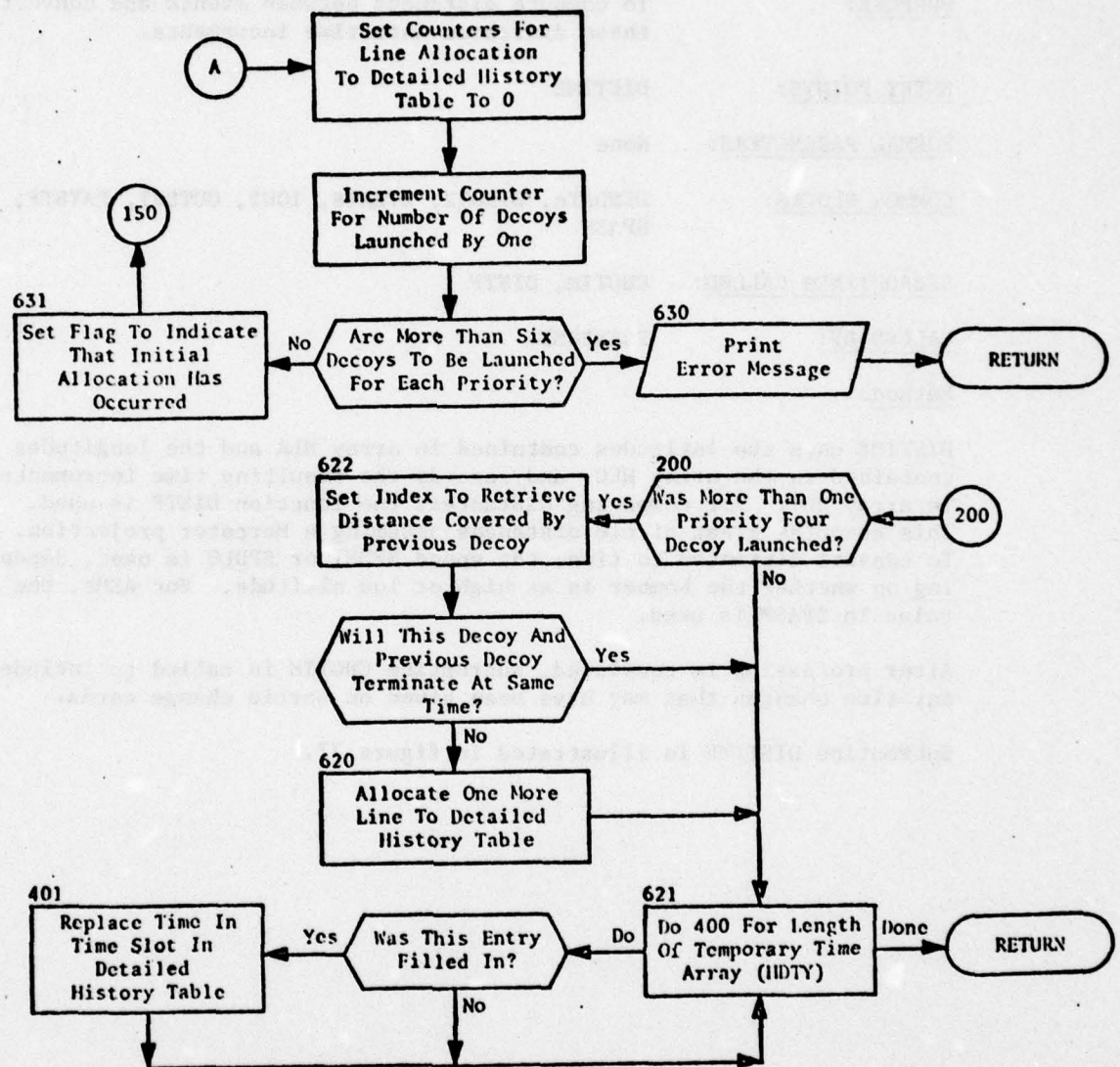


Figure 76. (Part 7 of 7)

4.8.5 Subroutine DISTIME

PURPOSE: To compute distances between events and convert these distances into time increments.

ENTRY POINTS: DISTIME

FORMAL PARAMETERS: None

COMMON BLOCKS: DINDATA, DINDT2, EVENTS, IOUT, OUTSRT, PAYSTF, SPASM

SUBROUTINES CALLED: CHGTIM, DISTF

CALLED BY: PLANBOMB

Method:

DISTIME uses the latitudes contained in array HLA and the longitudes contained in the array HLO, and records the resulting time increments in array HDT. For computing distances, the function DISTF is used. This computes great circle distances assuming a Mercator projection. To convert distances to time, the speed SPDHI or SPDLQ is used, depending on whether the bomber is at high or low altitude. For ASMs, the value in SPASM is used.

After processing is completed, subroutine CHGTIM is called to include any time changes that may have been given on sortie change cards.

Subroutine DISTIME is illustrated in figure 77.

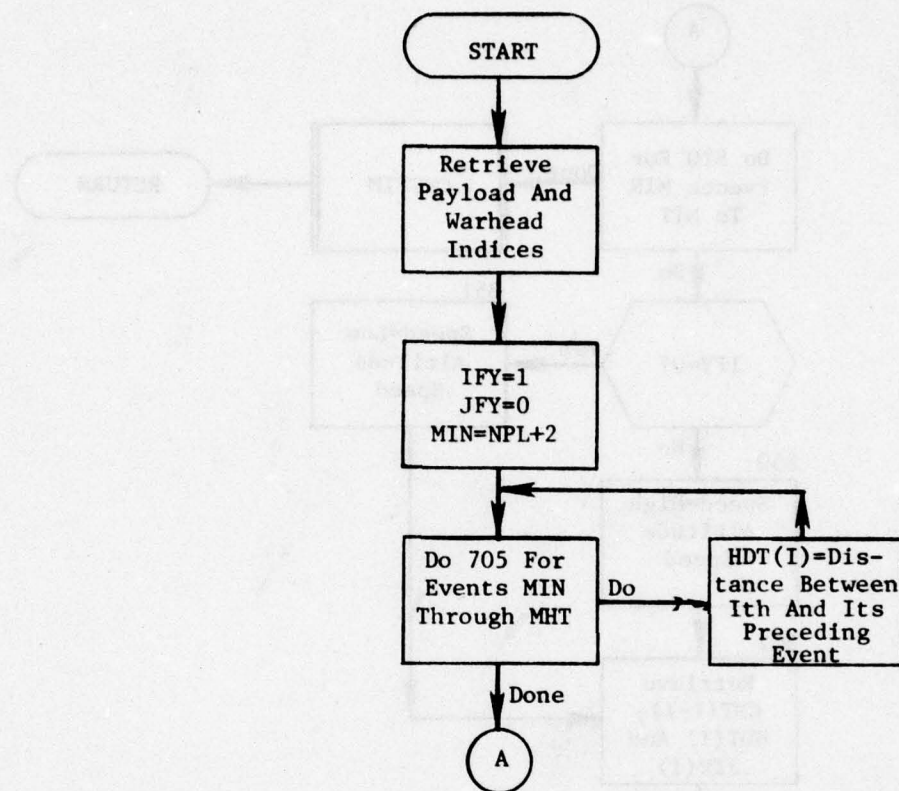


Figure 77. Subroutine DISTIME (Part 1 of 3)

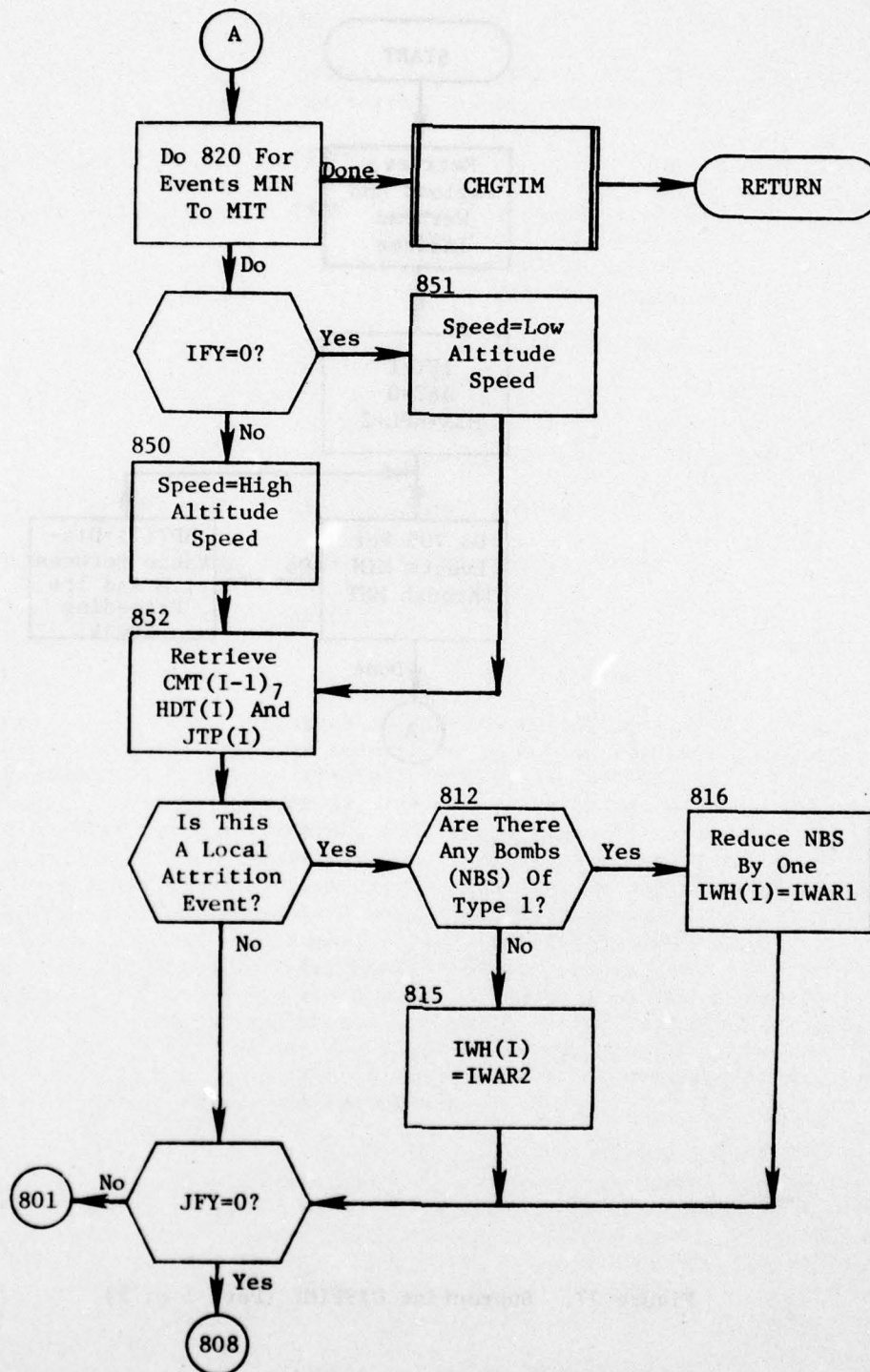


Figure 77. (Part 2 of 3)

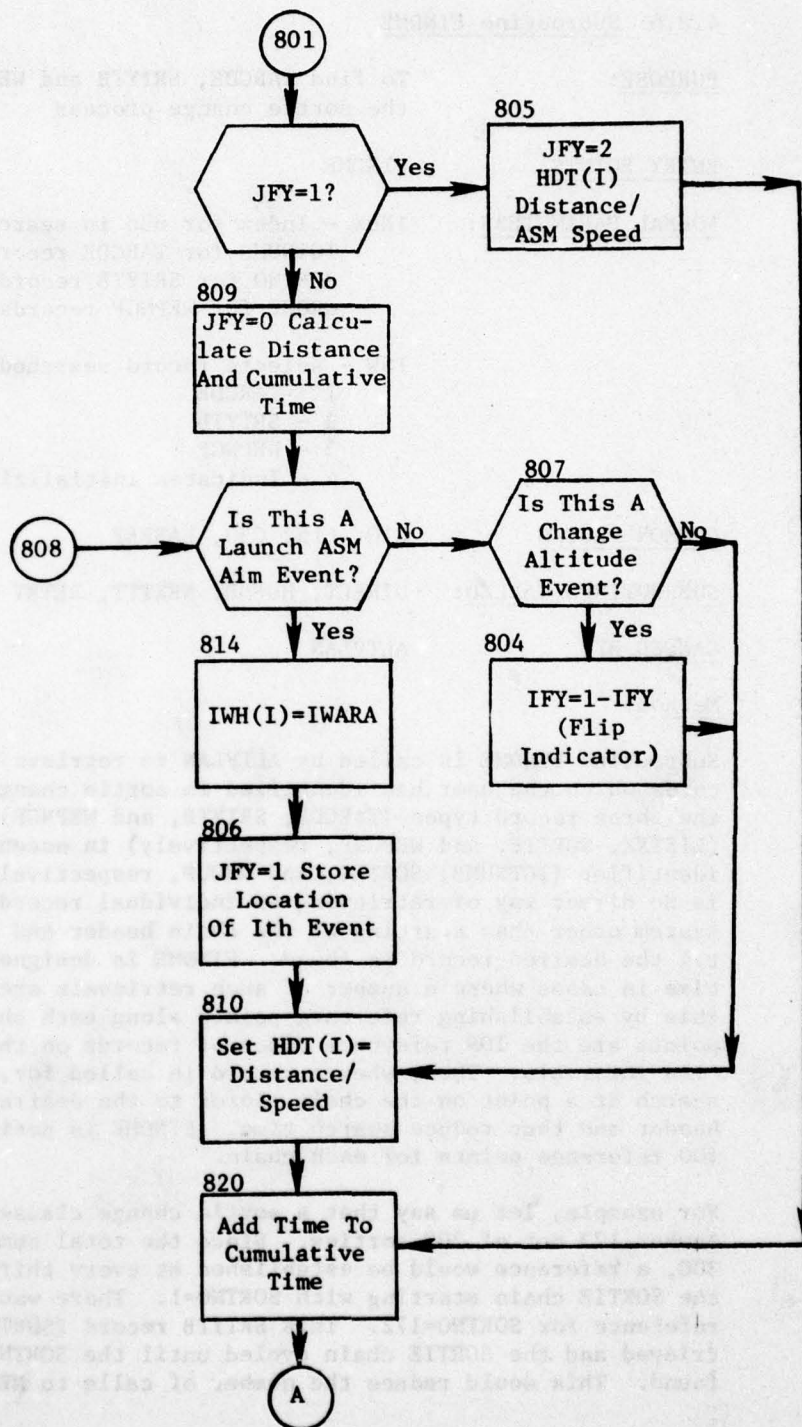


Figure 77. (Part 3 of 3)
409

4.8.6 Subroutine FINDME

PURPOSE: To find TARCDE, SRTYTB and WEPNGP records during the sortie change process

ENTRY POINTS: FINDME

FORMAL PARAMETERS: INDX - Index for use in search:
TGTNUMB for TARCDE records
SORTNO for SRTYTB records
GROUP for WEPNGP records

ISW - selects record searched for:
1 - TARCDE
2 - SRTYTB
3 - WEPNGP
4 - Indicates initializing call to subroutine

COMMON BLOCKS: C10, C15, C30, LASREF

SUBROUTINES CALLED: DIRECT, HDFND, NEXTTT, RETRV

CALLED BY: ALTPLAN

Method:

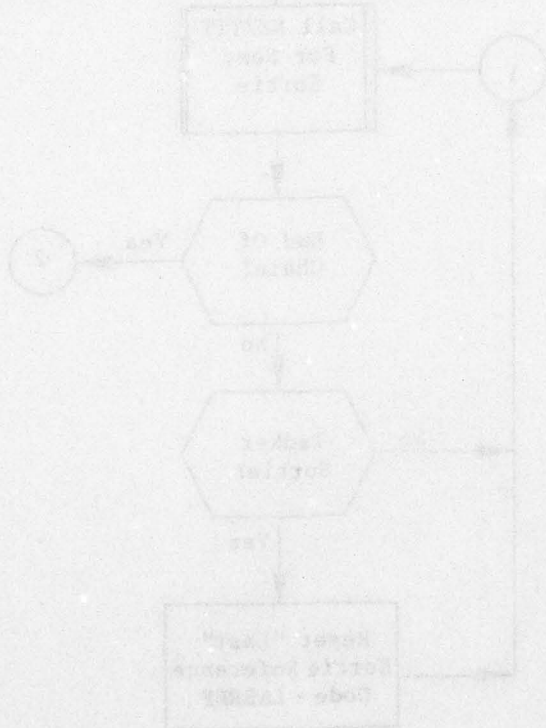
Subroutine FINDME is called by ALTPLAN to retrieve various types of records which the user has identified in sortie change clauses. Each of the three record types (TARCDE, SRTYTB, and WEPNGP) reside on a chain (LISTXX, SORTIE, and WEPGRP, respectively) in ascending order of their identifier (TGTNUMB, SORTNO, and GROUP, respectively). However, there is no direct way of retrieving an individual record built into the IDS system other than starting at the chain header and cycling the chain until the desired record is found. FINDME is designed to reduce processing time in cases where a number of such retrievals are called for. It does this by establishing reference points along each chain. These reference points are the IDS reference codes of records on the chains spaced at even intervals. Then, when a record is called for, FINDME can begin the search at a point on the chain closer to the desired record than the header and thus reduce search time. FINDME is designed to reserve up to 100 reference points for each chain.

For example, let us say that a sortie change clause referred to sortie number 173 out of 300 sorties. Since the total number of sorties is 300, a reference would be established at every third SRTYTB record on the SORTIE chain starting with SORTNO=1. There would, therefore, be a reference for SORTNO=172. This SRTYTB record (SORTNO=172) would be retrieved and the SORTIE chain cycled until the SORTNO=173 record was found. This would reduce the number of calls to NEXTTT from 173 to 1.

The first step in the FINDME process is the initialization call (ISW=4). This call determines the proper interval between reference records (IBK) by determining the number of records on the chain and dividing by 100. An ancillary effect is the resetting of LASREF (reference code of the last nontanker sortie) for any missile sorties added by ACARD clauses. After this interval is established, the first record of each of the chains is retrieved and saved as the first reference record.

Then for each call, the closest potential reference record is determined. If it has already been found FINDME retrieves it immediately. If not, the closest reference record previously found is retrieved (on the first call for any chain this would be the first reference record) and the chain is cycled until the potential reference record is found. During this cycling process, any intervening reference records have their IDS reference codes saved. Once the desired reference record is obtained, the chain is cycled until the record desired in the subroutine call is found.

Subroutine FINDME is illustrated in figure 78.



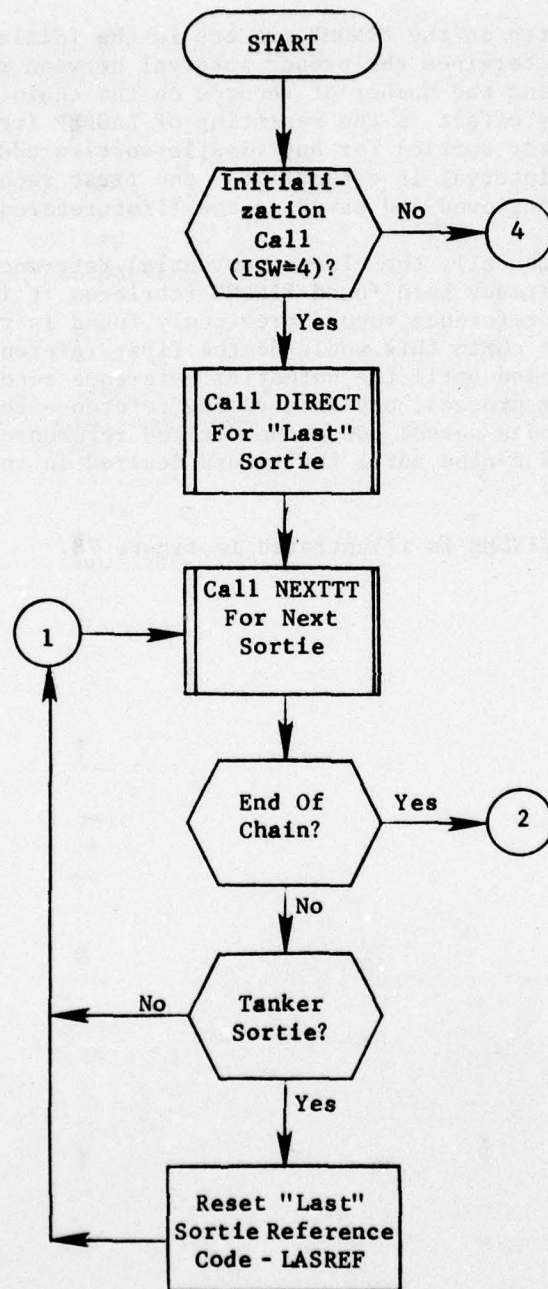


Figure 78. Subroutine FINDME (Part 1 of 4)

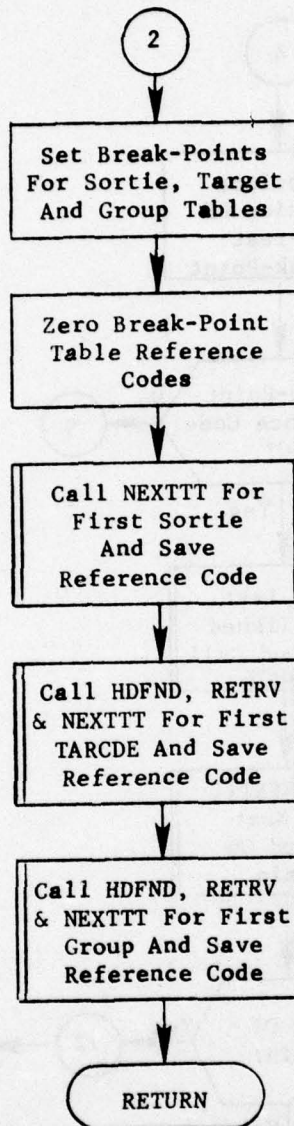


Figure 78. (Part 2 of 4)

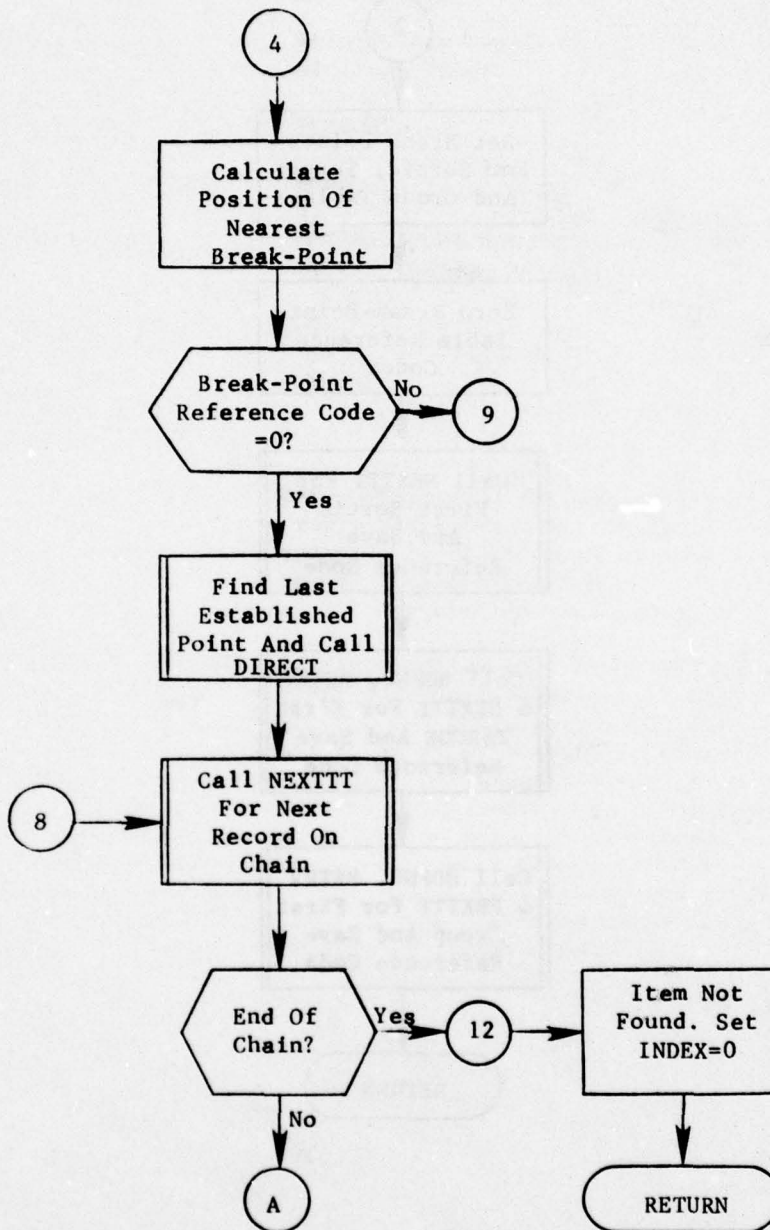


Figure 78. (Part 3 of 4)

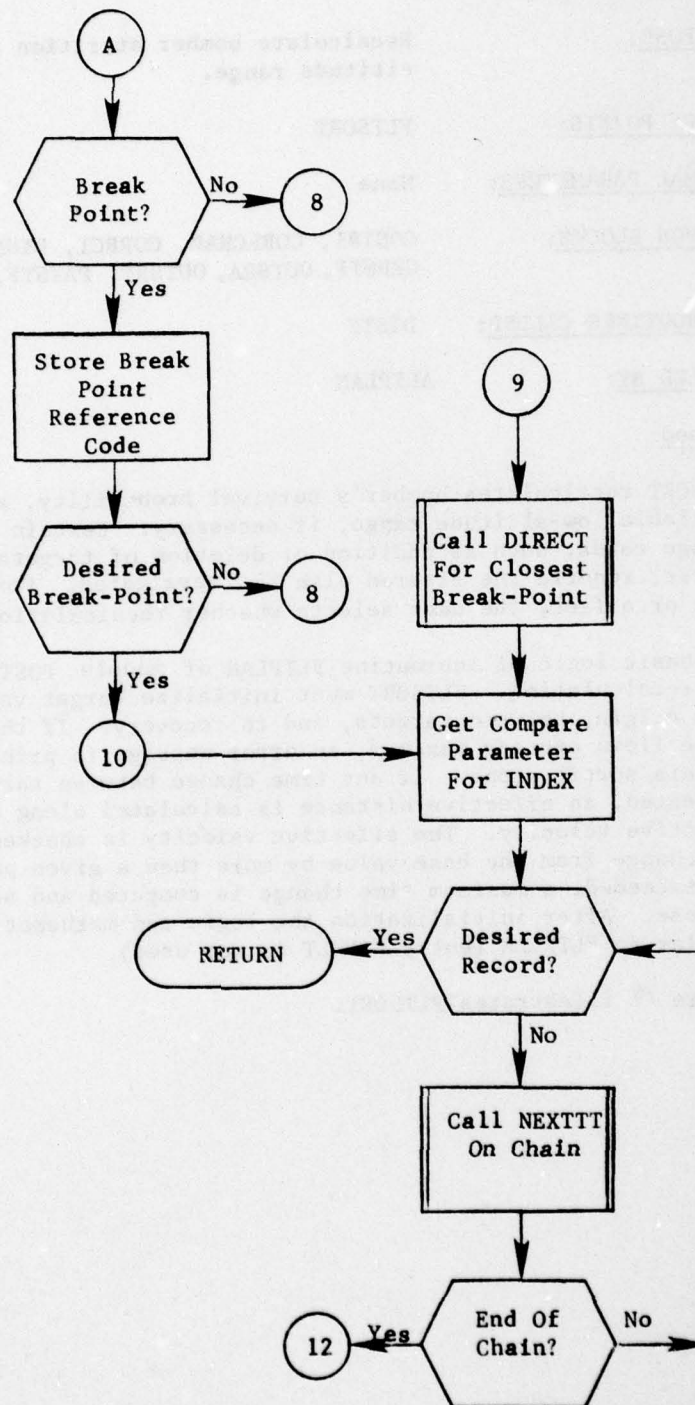


Figure 78. (Part 4 of 4)

4.8.7 Subroutine FLTSORT

PURPOSE: Recalculate bomber attrition and available low altitude range.

ENTRY POINTS: FLTSORT

FORMAL PARAMETERS: None

COMMON BLOCKS: CONTR1, CORRCHAR, CORRC1, DINDATA, DPENREF, GRPSTF, OUTSRA, OUTSRT, PAYSTF, TYPSTF, WHDSTF

SUBROUTINES CALLED: DISTF

CALLED BY: ALTPLAN

Method:

FLTSORT recalculates bomber's survival probability, attrition, and available low-altitude range, if necessary. Certain changes on sortie change cards, such as addition or deletion of targets from the original sortie, require the altered plan be reevaluated. For minor changes, time or offset, the user selects whether recalculation is desired.

The basic logic of subroutine FLTPLAN of module POSTALOC is used for the recalculation. FLTSORT must initialize target values and distances from origin, between targets, and to recovery. If the total distance to be flown exceeds maximum, an error message is printed and processing of this sortie stops. If any time change between targets has been requested, an effective distance is calculated along with a corresponding effective velocity. The effective velocity is checked to ensure it does not change from the base value by more than a given percent. If limits are exceeded, a maximum time change is computed and stored as the value for use. After initialization the logic and mathematical technique is similar to FLTPLAN (entry FINFLT is not used).

Figure 79 illustrates FLTSORT.

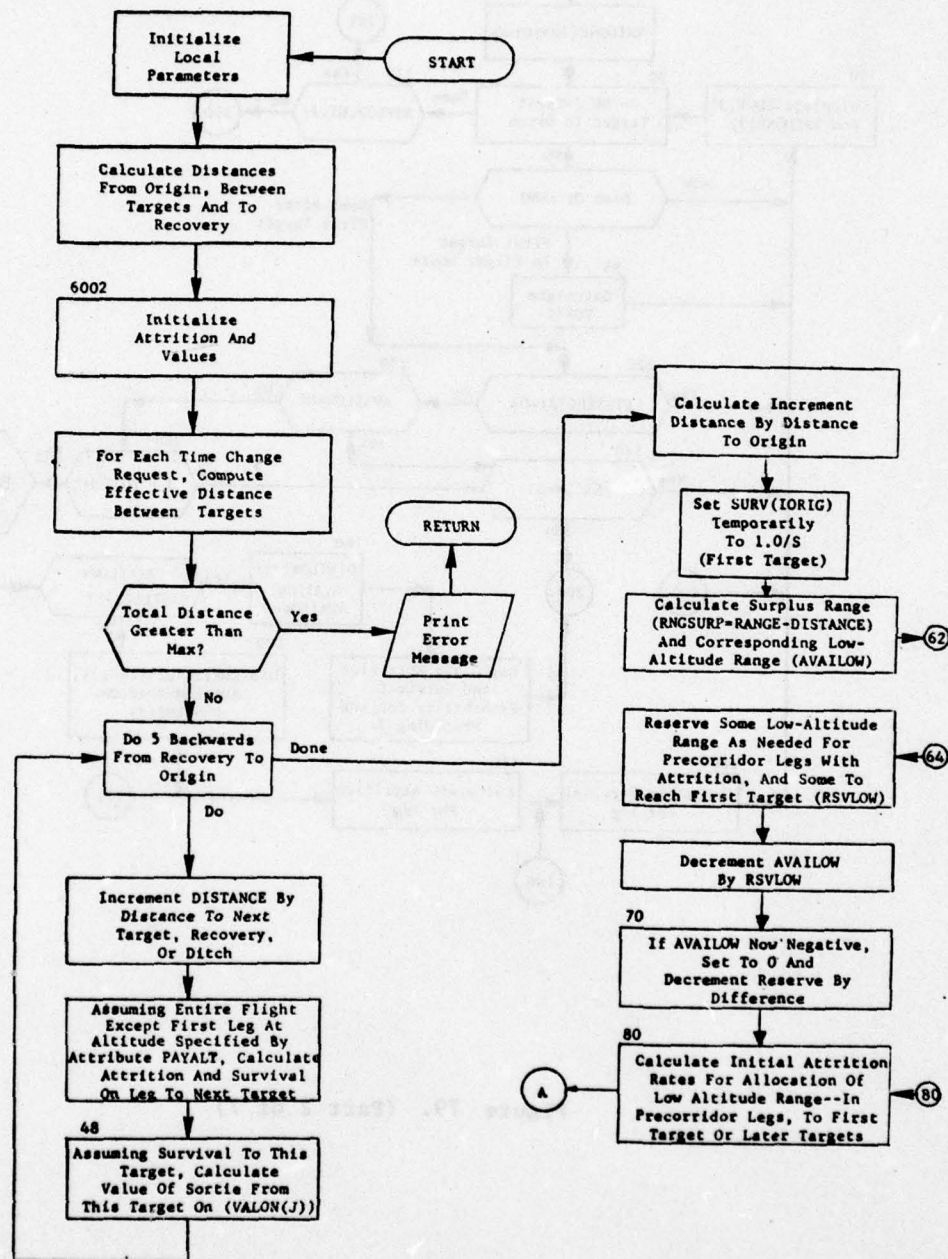


Figure 79. Subroutine FLTSORT
(Part 1 of 7)

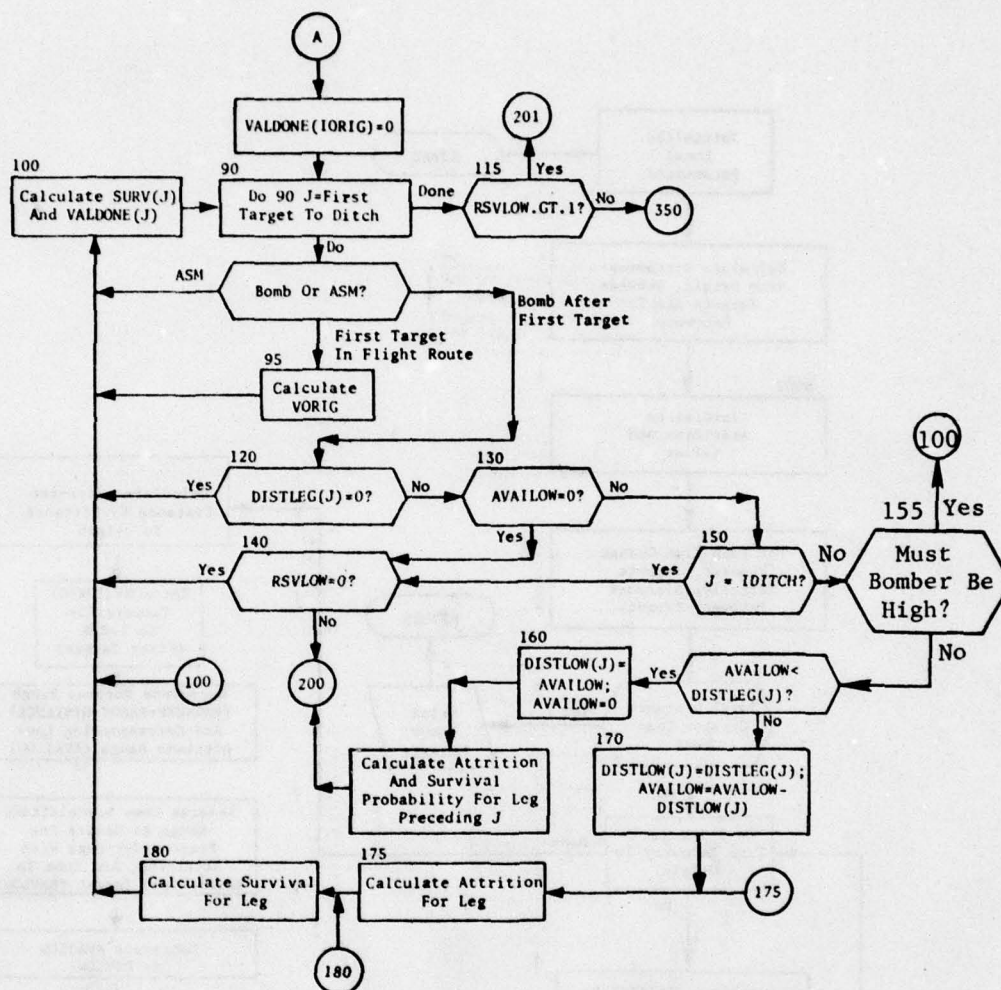


Figure 79. (Part 2 of 7)

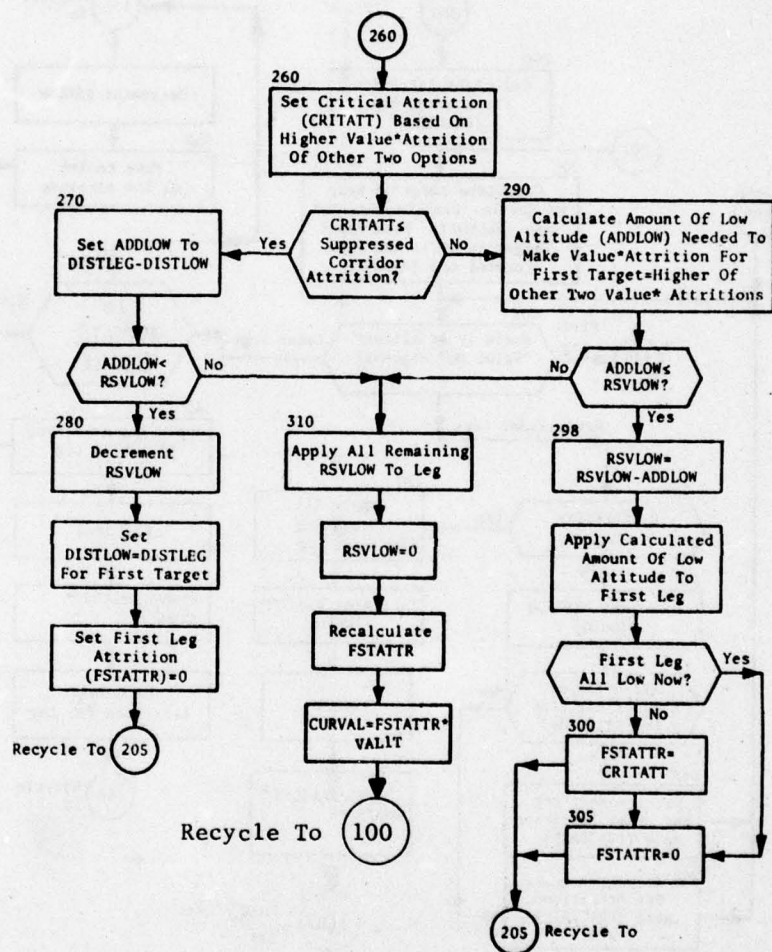


Figure 79. (Part 4 of 7)

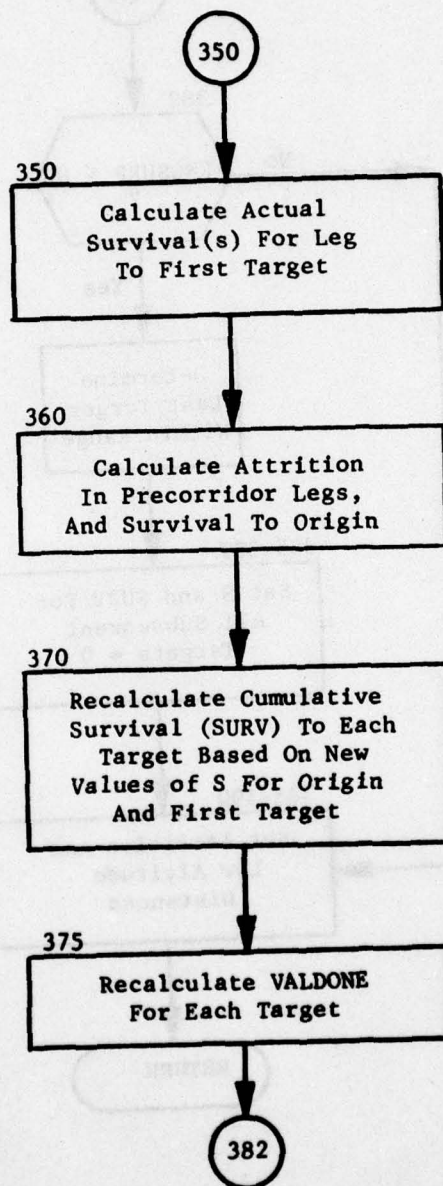


Figure 79. (Part 5 of 7)

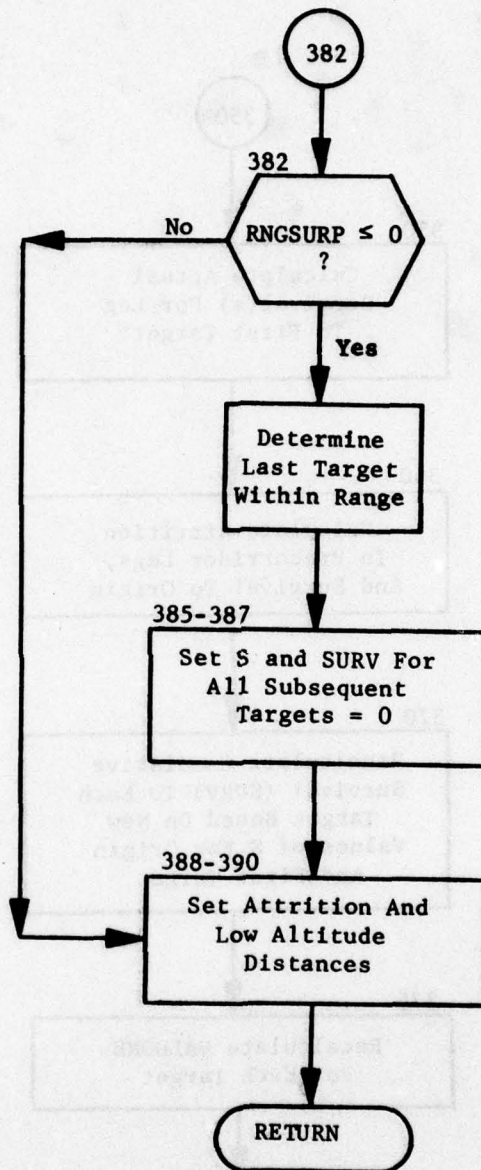


Figure 79. (Part 6 of 7)

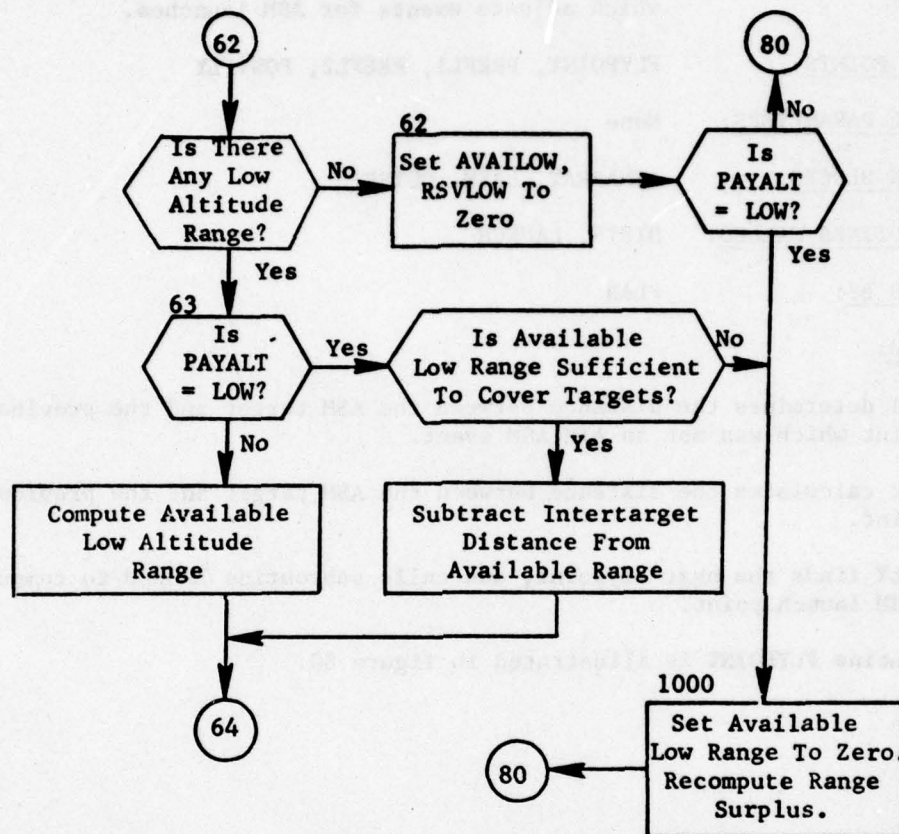


Figure 79. (Part 7 of 7)

4.8.8 Subroutine FLYPOINT

PURPOSE: FLYPOINT is an integral part of block 40 in PLAN which adjusts events for ASM launches.

ENTRY POINTS: FLYPOINT, PREFL1, PREFL2, POSTFLY

FORMAL PARAMETERS: None

COMMON BLOCKS: ASMARRAY, LASM, OUTSRT

SUBROUTINES CALLED: DISTF, LAUNCH

CALLED BY: PLAN

Method:

PREFL1 determines the distance between the ASM target and the previous flypoint which was not an AIM ASM event.

PREFL2 calculates the distance between the ASM target and the previous flypoint.

POSTFLY finds the next flypoint, and calls subroutine LAUNCH to compute the ASM launch point.

Subroutine FLYPOINT is illustrated in figure 80.

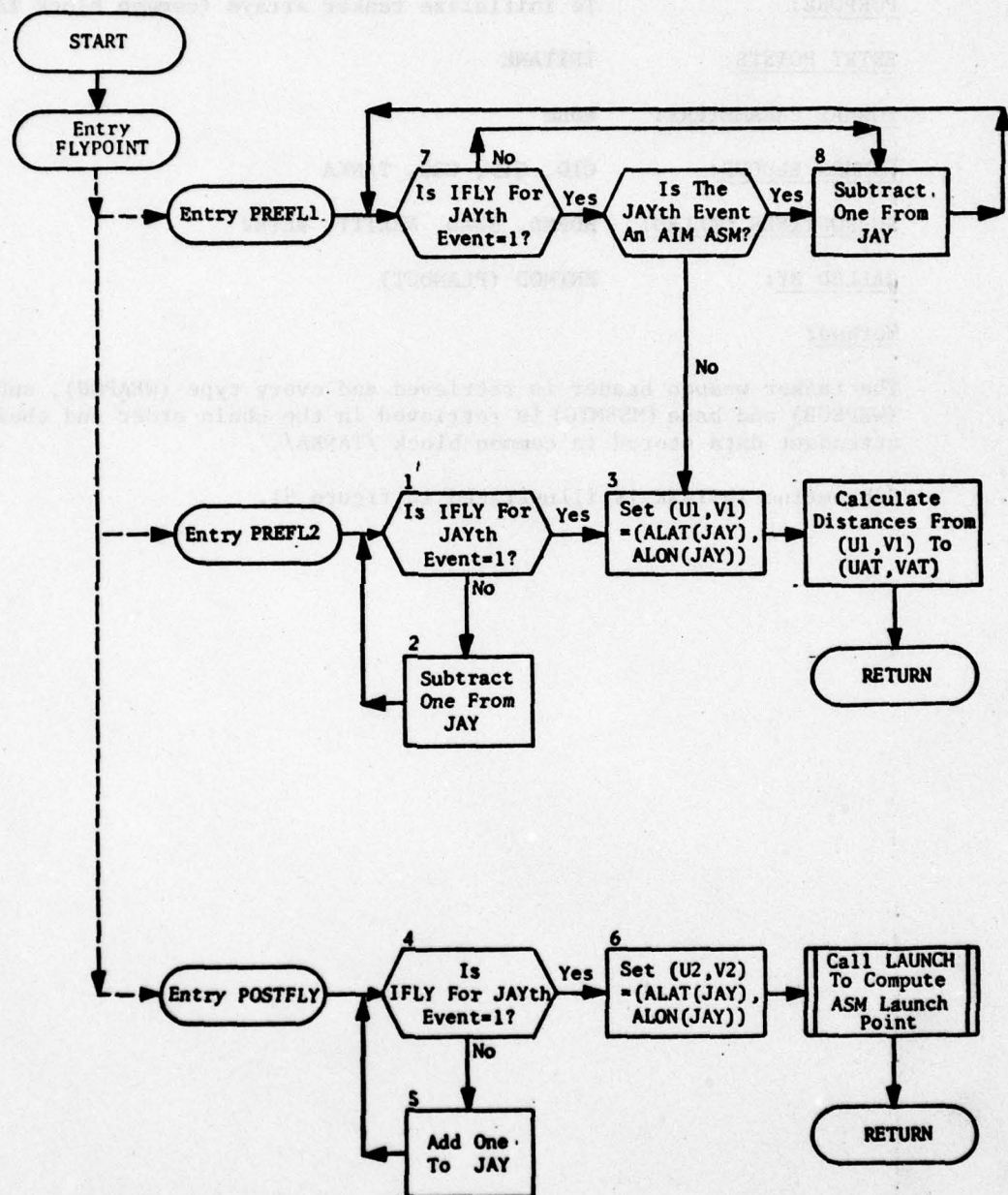


Figure 80. Subroutine FLYPOINT

4.8.9 Subroutine INITANK

PURPOSE: To initialize tanker arrays (common block TANKA)

ENTRY POINTS: INITANK

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, TANKA

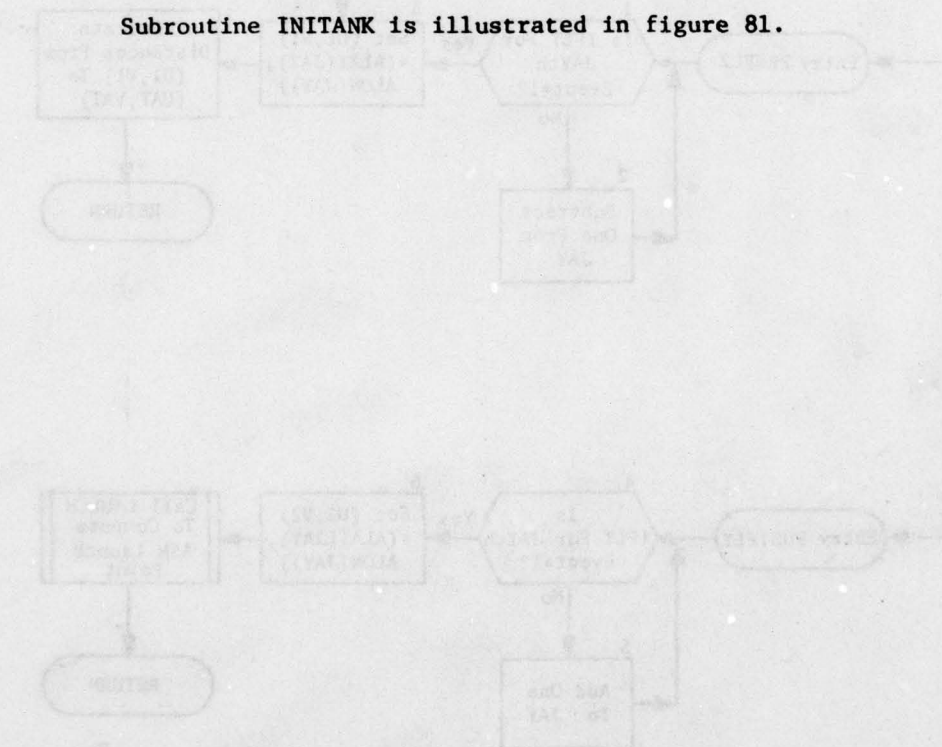
SUBROUTINES CALLED: HDFND, HEAD, NEXTTT, RETRV

CALLED BY: ENTMOD (PLANOUT)

Method:

The tanker weapon header is retrieved and every type (WEAPON), subtype (WEPSUB) and base (MSBMTG) is retrieved in the chain order and their attendant data stored in common block /TANKA/.

Subroutine INITANK is illustrated in figure 81.



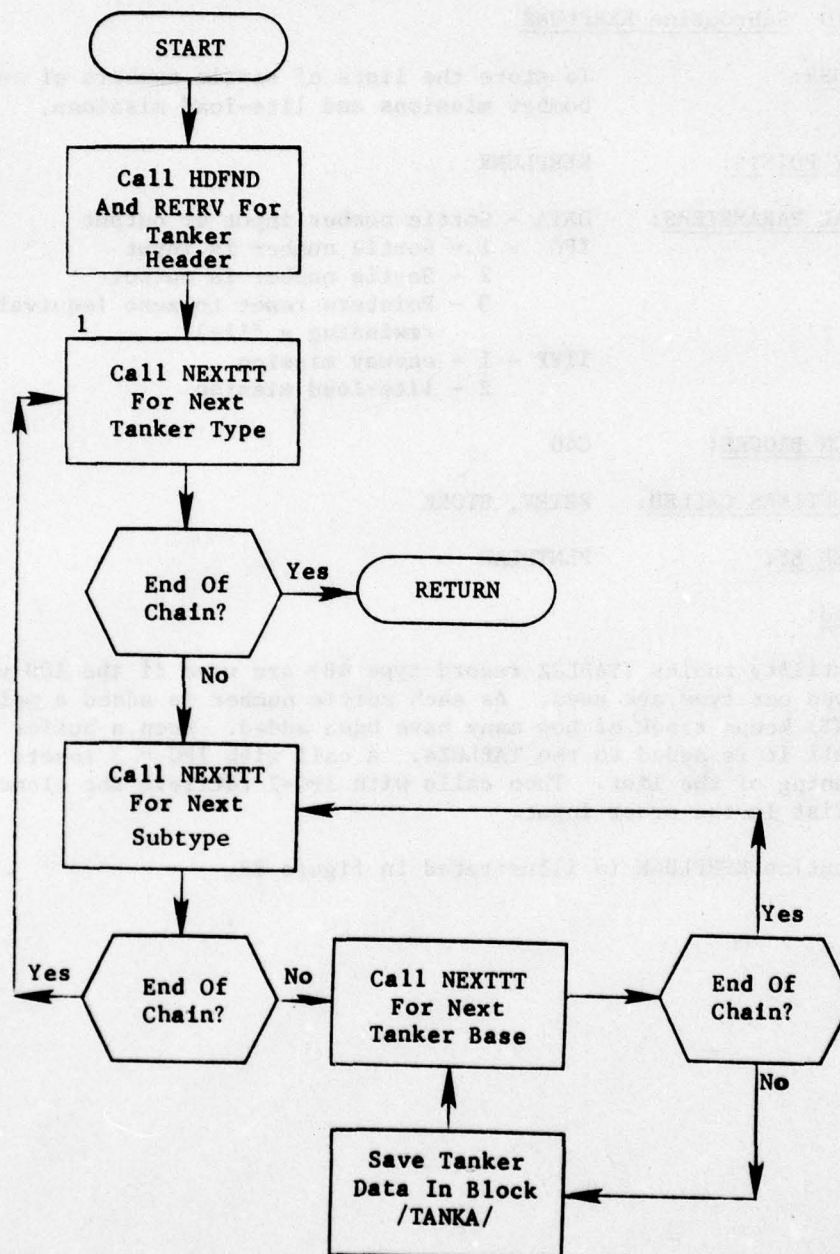


Figure 81. Subroutine INITANK

4.8.10 Subroutine KERPLUNK

PURPOSE: To store the lists of sortie numbers of oneway bomber missions and lite-load missions.

ENTRY POINTS: KERPLUNK

FORMAL PARAMETERS: DATA - Sortie number input or output
IPG - 1 - Sortie number is input
2 - Sortie number is output
3 - Pointers reset to zero (equivalent to rewinding a file)
ITYP - 1 - oneway mission
2 - lite-load mission

COMMON BLOCKS: C40

SUBROUTINES CALLED: RETRV, STORE

CALLED BY: PLNTPLAN

Method:

The utility tables (TABLEZ record type 48) are used if the 100 words allowed per type are used. As each sortie number is added a pointer (COUNT) keeps track of how many have been added. When a buffer (BUFF) is full it is added to the TABLEZs. A call with IPG = 3 resets to the beginning of the list. Then calls with IPG=2 retrieve the elements of the list in the order input.

Subroutine KERPLUNK is illustrated in figure 82.

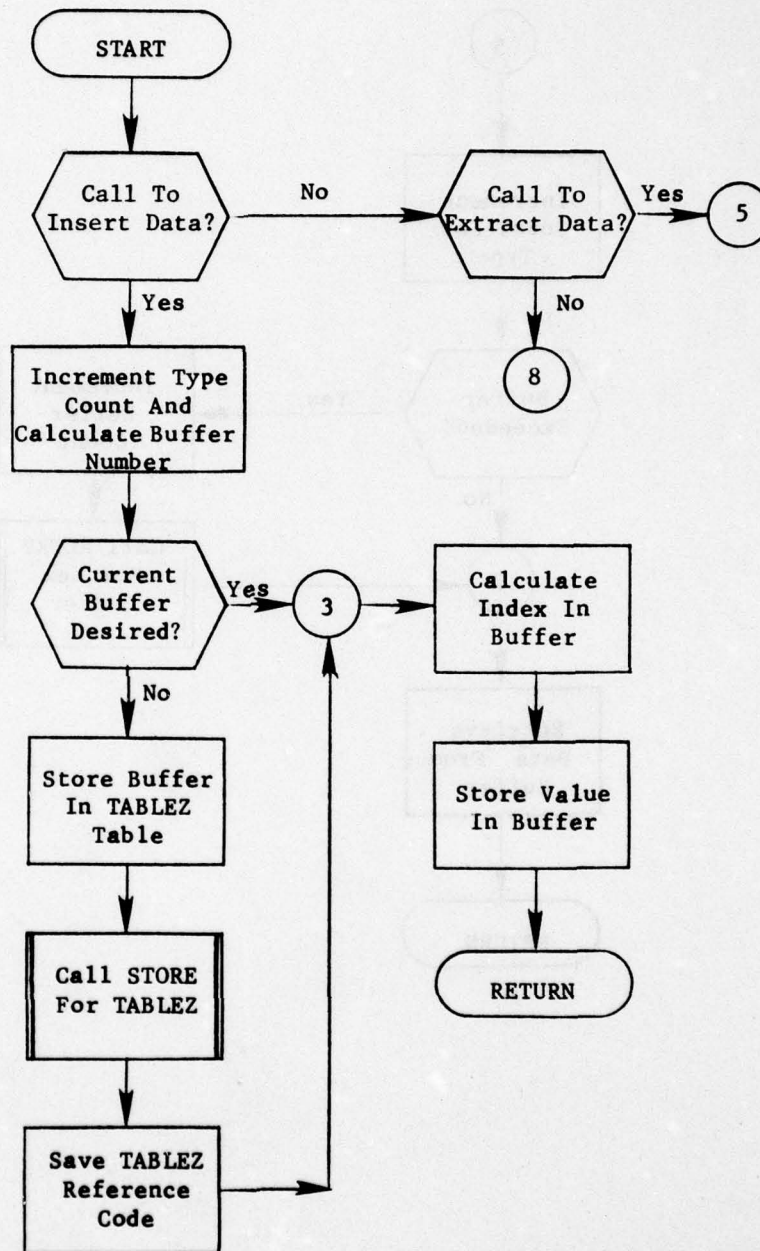


Figure 82. Subroutine KERPLUNK (Part 1 of 3)

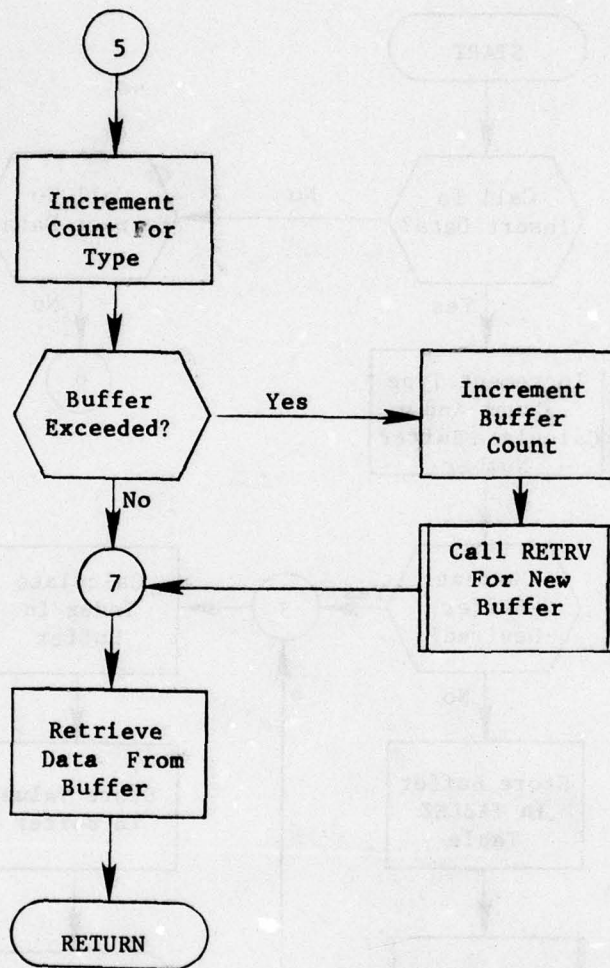


Figure 82. (Part 2 of 3)

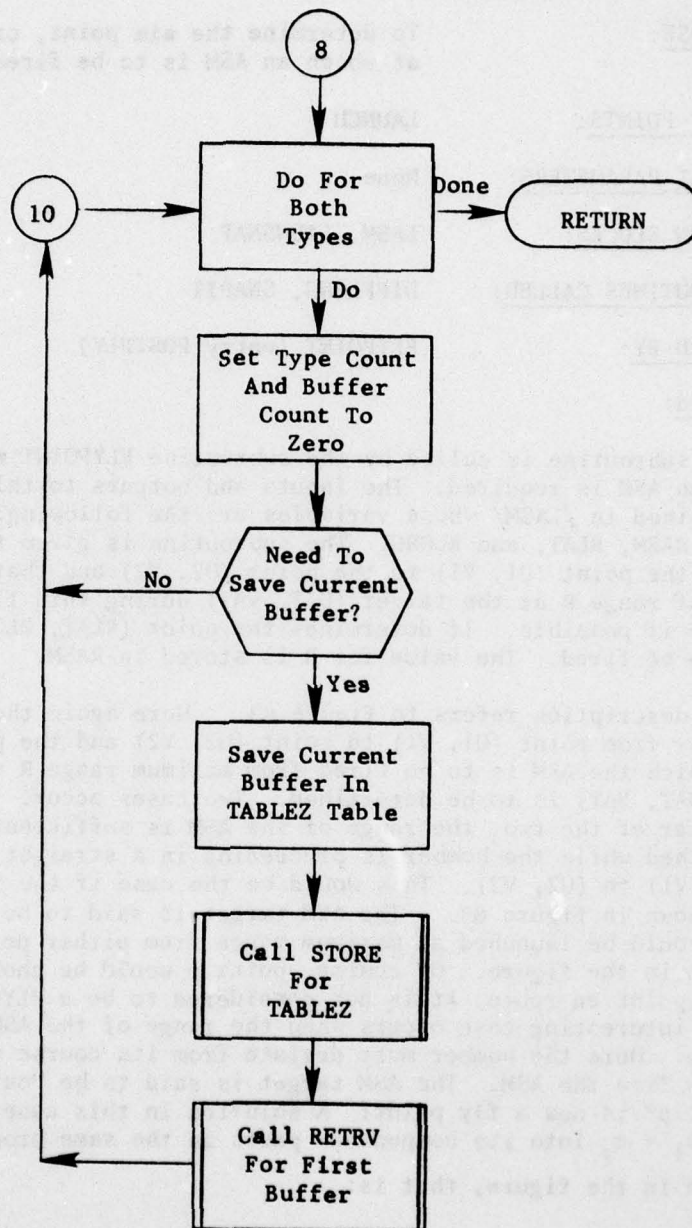


Figure 82. (Part 3 of 3)

4.8.11 Subroutine LAUNCH

PURPOSE: To determine the aim point, or launch point, at which an ASM is to be fired.

ENTRY POINTS: LAUNCH

FORMAL PARAMETERS: None

COMMON BLOCKS: LASM, LAUNSNAP

SUBROUTINES CALLED: DIFFLONG, SNAPIT

CALLED BY: FLYPOINT (entry POSTFLY)

Method:

This subroutine is called by the subroutine FLYPOINT whenever the aim point for an ASM is required. The inputs and outputs to this subroutine are all contained in /LASM/ whose variables are the following: U1, V1, U2, V2, UAT, VAT, RASM, RLAT, and RLONG. The subroutine is given that a bomber is flying from the point (U1, V1) to the point (U2, V2) and that it is to fire an ASM of range R at the target (UAT, VAT) during this flight, at maximum range if possible. It determines the point (RLAT, RLONG) at which the ASM is to be fired. The value for R is stored in RASM.

This description refers to figure 83. Here again the bomber is assumed to fly from point (U1, V1) to point (U2, V2) and the point (RLAT, RLONG) at which the ASM is to be fired from maximum range R to a target located at (UAT, VAT) is to be determined. Two cases occur. In the first and simpler of the two, the range of the ASM is sufficient so that it may be launched while the bomber is proceeding in a straight line path from (U1, V1) to (U2, V2). This would be the case if the range of the ASM were R' shown in figure 83. The ASM target is said to be "in range". The ASM could be launched at maximum range from either point p or point p' shown in the figure. Of course, point p would be chosen. Since point p is a point en route, it is not considered to be a FLYPOINT. The second and more interesting case occurs when the range of the ASM is equal to R as shown. Here the bomber must deviate from its course and fly to the point p" to fire the ASM. The ASM target is said to be "out of range", and the point p" is now a fly point. A solution in this case is to divide the angle $\theta = \theta_1 + \theta_2$ into its components parts in the same proportion as D1 and D2 shown in the figure, that is:

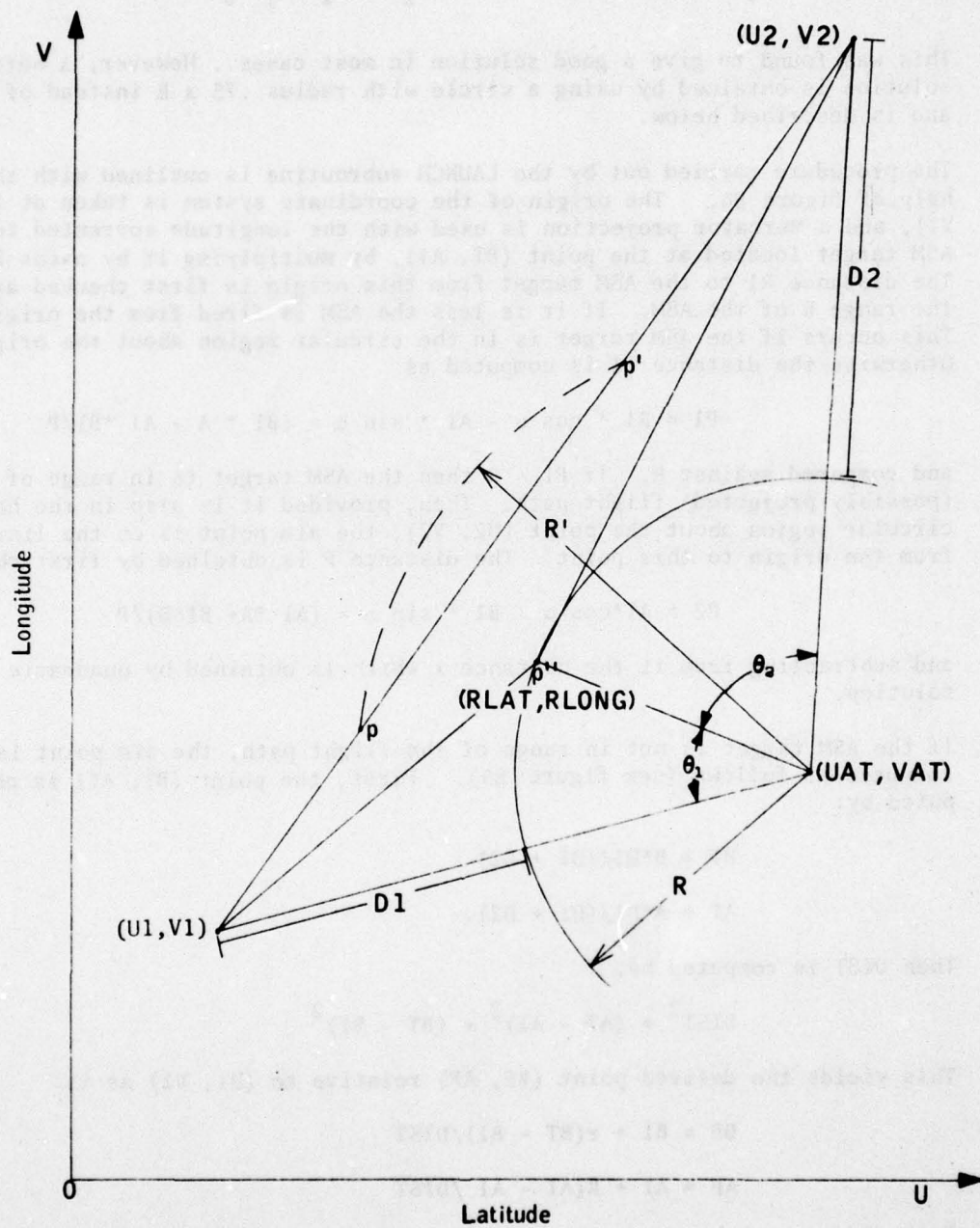


Figure 83. Determination of ASM Aim Point

$$\theta_1 = \theta D_1 / (D_1 + D_2)$$

$$\theta_2 = \theta D_2 / (D_1 + D_2)$$

This was found to give a good solution in most cases.. However, a better solution is obtained by using a circle with radius $.75 \times R$ instead of R , and is described below.

The procedure carried out by the LAUNCH subroutine is outlined with the help of figure 84. The origin of the coordinate system is taken at $(U1, V1)$, and a Mercator projection is used with the longitude corrected to the ASM target located at the point $(B1, A1)$, by multiplying it by $\alpha = \cos B1$. The distance $R1$ to the ASM target from this origin is first checked against the range R of the ASM. If it is less the ASM is fired from the origin. This occurs if the ASM target is in the circular region about the origin. Otherwise the distance $P1$ is computed as

$$P1 = B1 * \cos \alpha - A1 * \sin \alpha = (B1 * A - A1 * B) / P$$

and compared against R . If $P1 < R$ then the ASM target is in range of the (possibly projected) flight path. Then, provided it is also in the half-circular region about the point $(U2, V2)$, the aim point is on the line from the origin to this point. The distance F is obtained by first obtaining

$$P2 = A1 * \cos \alpha + B1 * \sin \alpha = (A1 * A + B1 * B) / P$$

and subtracting from it the distance x which is obtained by quadratic solution.

If the ASM target is not in range of the flight path, the aim point is computed as follows (see figure 85). First, the point (BT, AT) is computed by:

$$BT = B * D1 / (D1 + D2)$$

$$AT = A * D1 / (D1 + D2).$$

Then $DIST$ is computed by:

$$DIST^2 = (AT - A1)^2 + (BT - B1)^2.$$

This yields the desired point (BF, AF) relative to $(U1, V1)$ as is:

$$BF = B1 + r(BT - B1) / DIST$$

$$AF = A1 + R(AT - A1) / DIST$$

from which are obtained $RLAT = U1 + BF$ and $RLONG = V1 + AF/\alpha$.

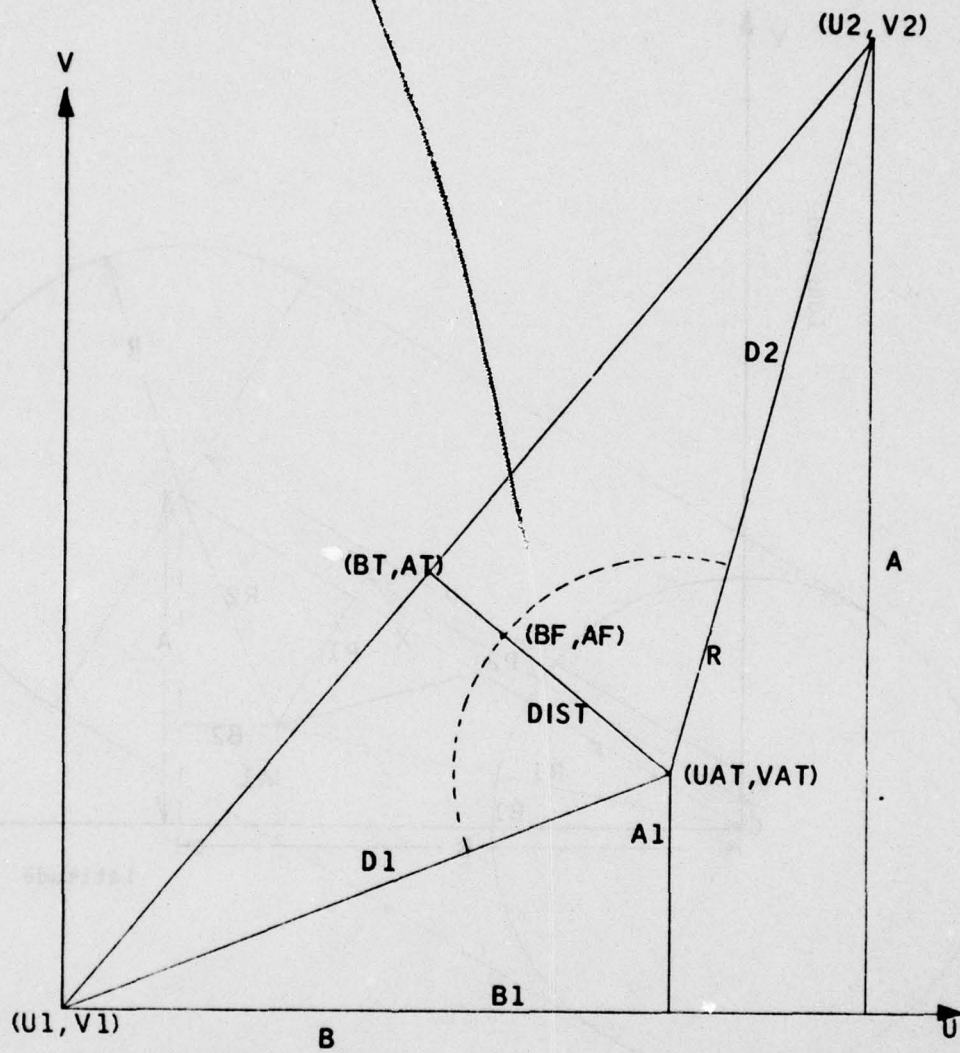


Figure 85. Computation of Flight Path Aim Point

The flowchart for LAUNCH is given in figure 86. In comparing it with the above description, it is useful to note that, in the program, quantities are squared for comparison purposes. Thus $(R1)^2 = R1SQ$, $(R2)^2 = R2SQ$, $p^2 = PATHSQ$, and $(P1)^2 = B1SQ$.

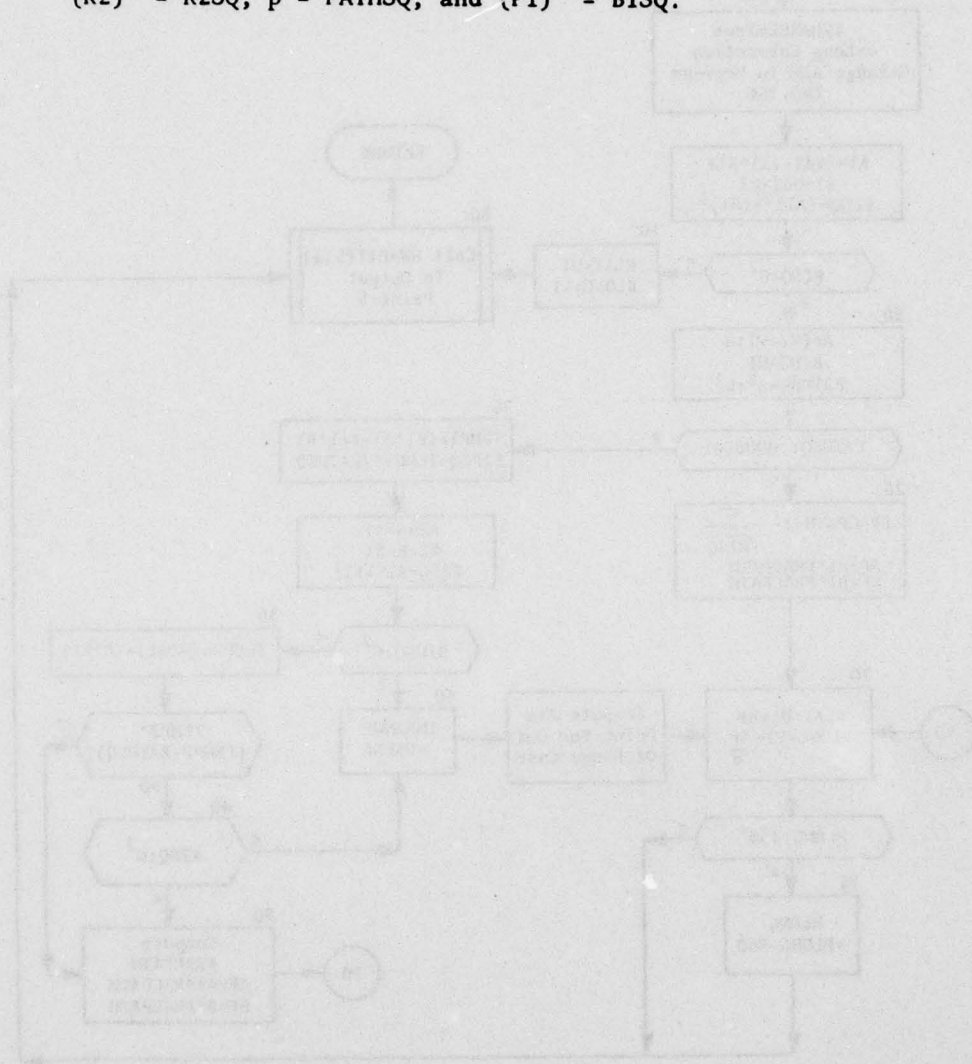


Figure 86. LAUNCH Program Flowchart

4.8.12 Subroutine LNCHDATA

PURPOSE: Read in missile timing data.

ENTRY POINTS: LNCHDATA

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, CALLSW, OOPS, TIMELINE, TYPSTF, ZEES

SUBROUTINES CALLED: ABORT, DISTF, INSGET, SLOG, XLL

CALLED BY: ALTPLAN, PLNTPLAN

Method:

This subroutine is best understood by reference to figure 87 as most of it is taken up with error analysis of the MISTME and MSLCOR clauses. For each MISTME clause, the endpoints are converted to decimal degrees by XLL and the length and crossproduct coefficients are stored in common /TIMELINE/. The data from the MSLCOR clauses is also stored in /TIMELINE/.

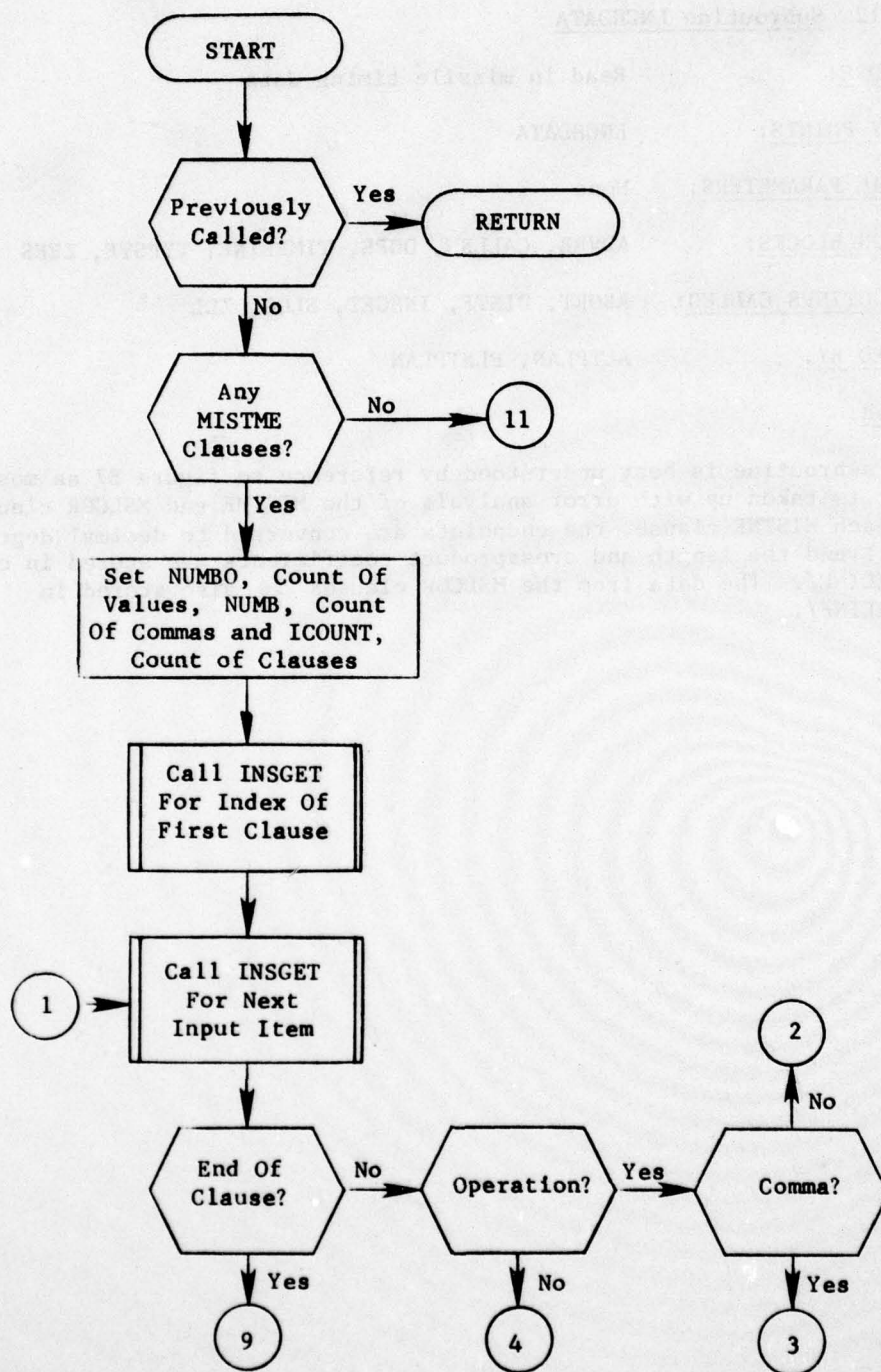


Figure 87. Subroutine LNCHDATA (Part 1 of 9)

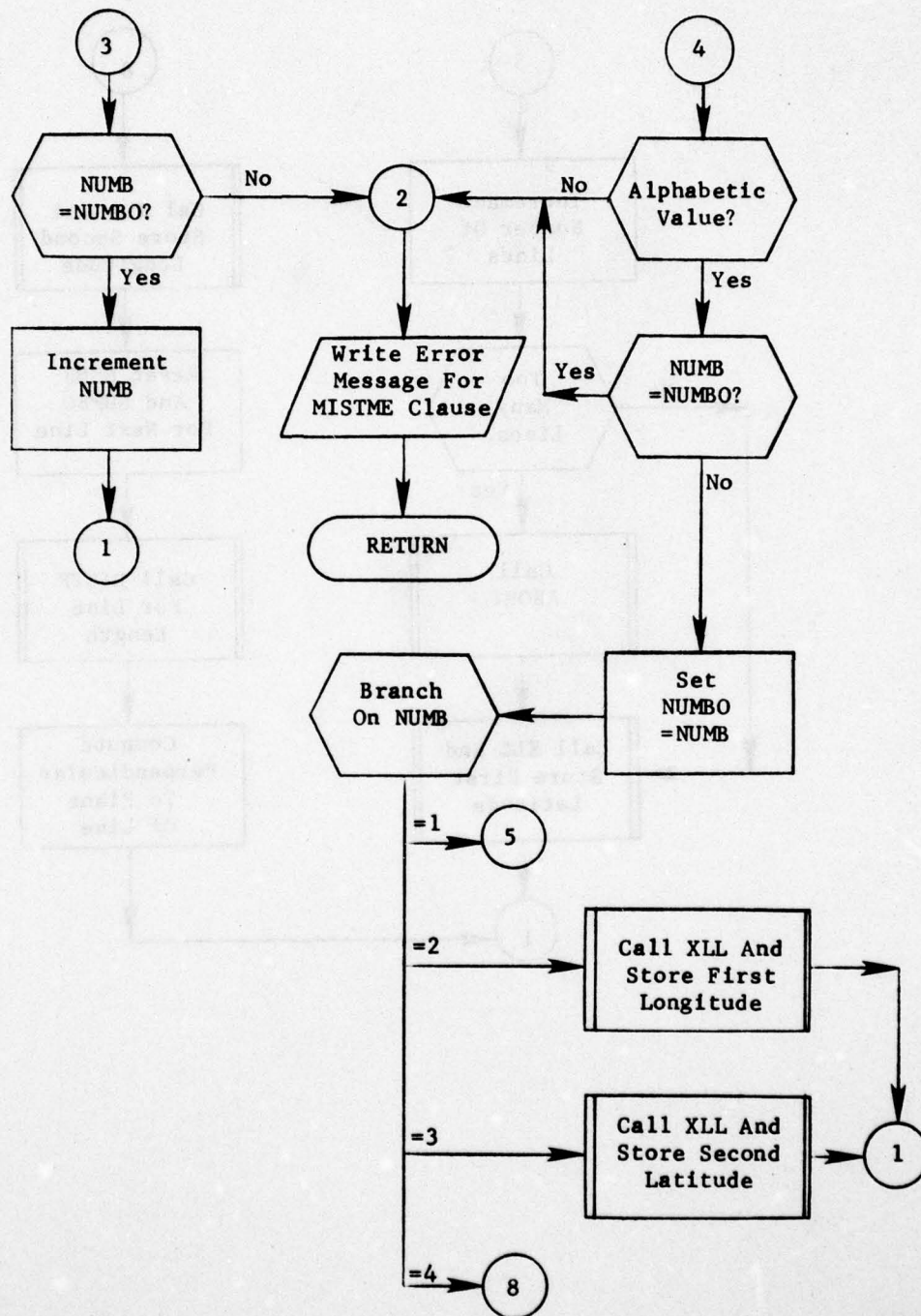


Figure 87. (Part 2 of 9)

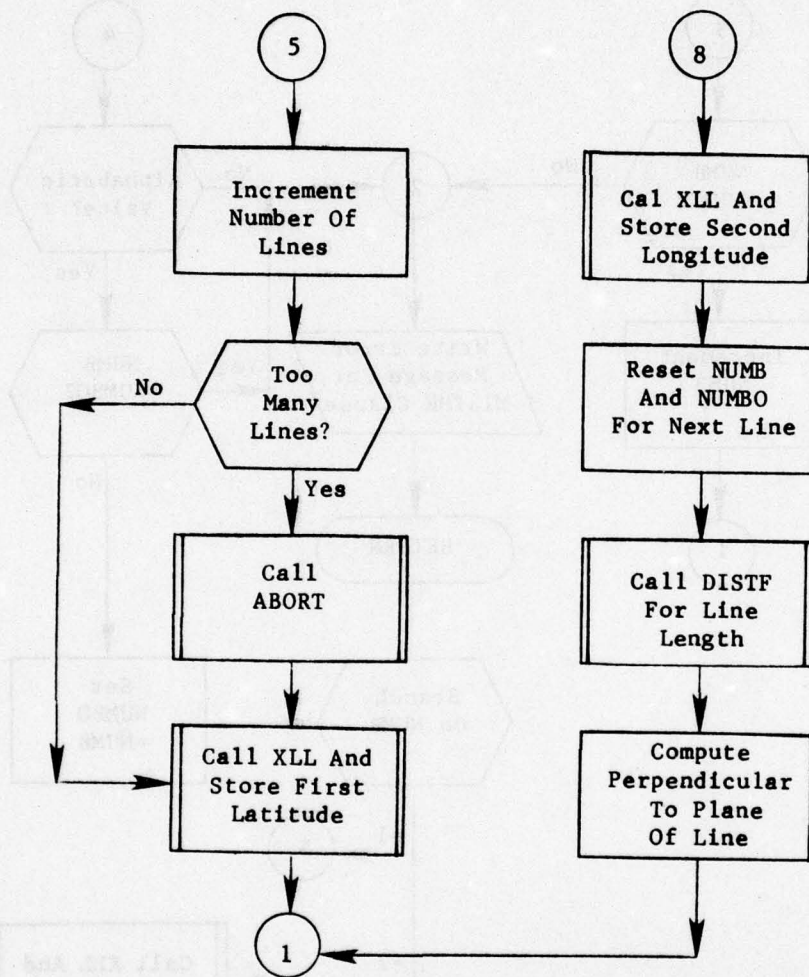


Figure 87. (Part 3 of 9)

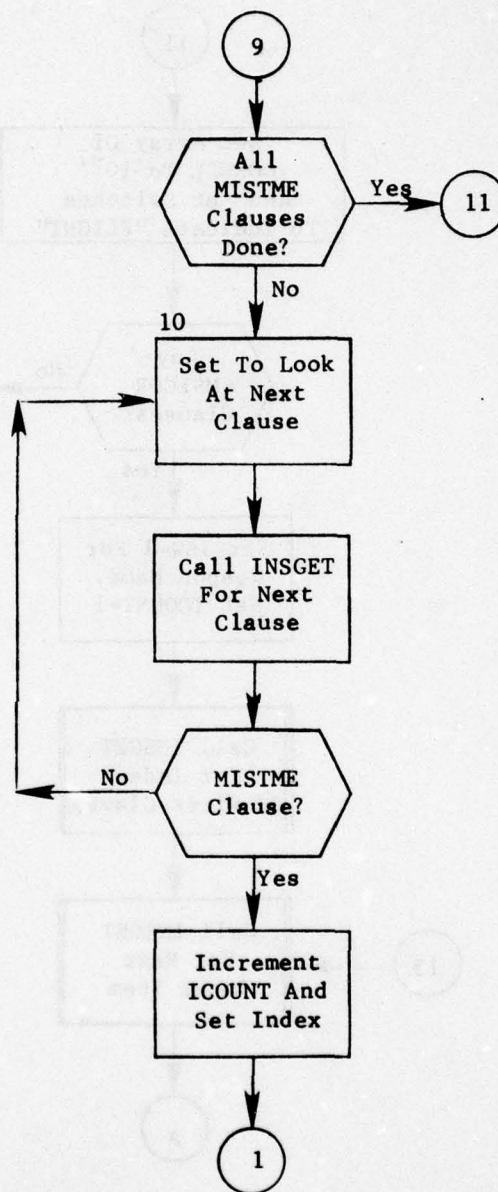


Figure 87. (Part 4 of 9)

AD-A058 406

COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK). VOLU--ETC(U)
APR 78 D J SANDERS, P F MAYKRANTZ, J M HERRON

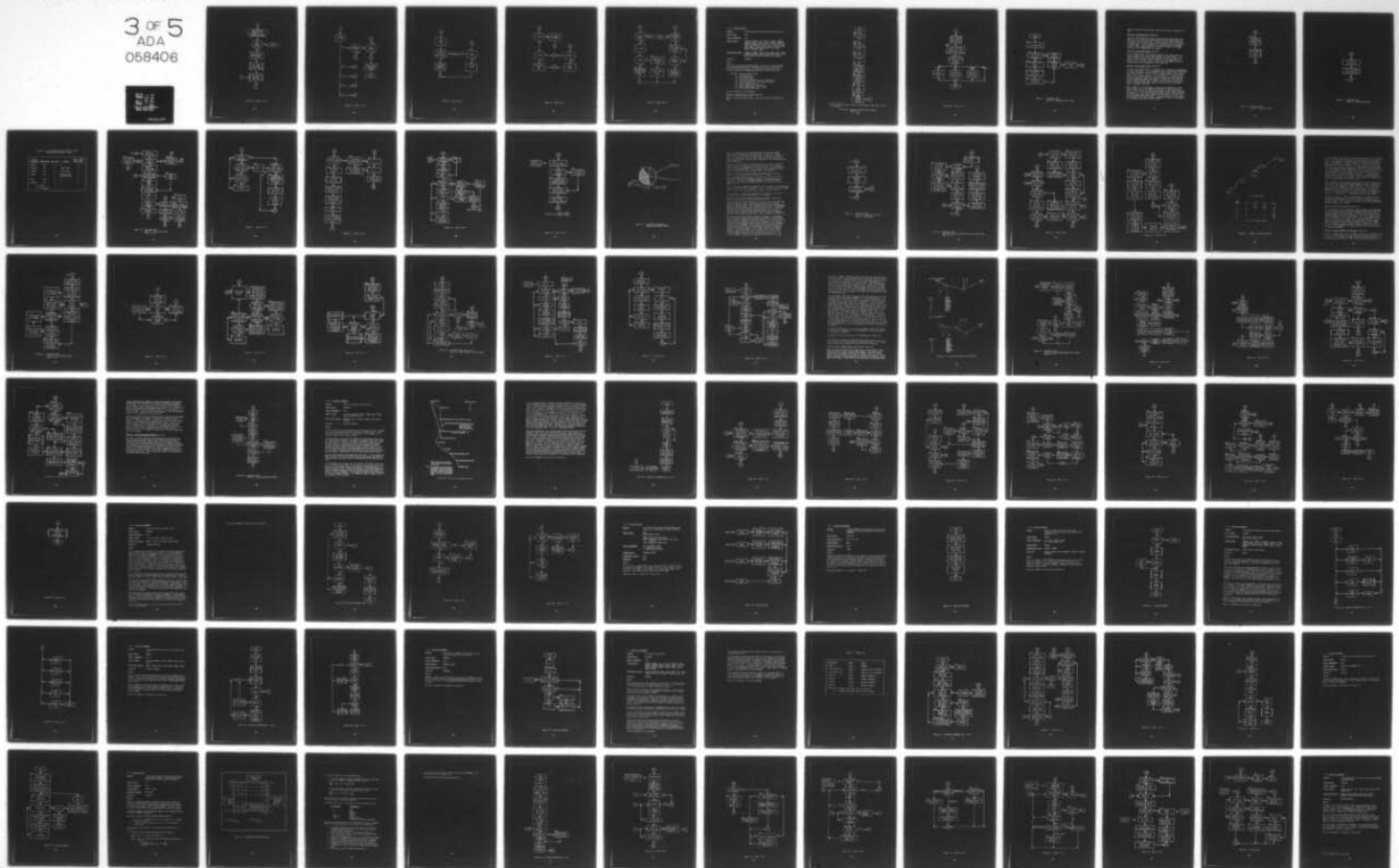
F/G 15/7

UNCLASSIFIED

CCTC-CSM-MM-9-77-VOL-4-PT SBIE-AD-E100 085

NL

3 OF 5
ADA
058406



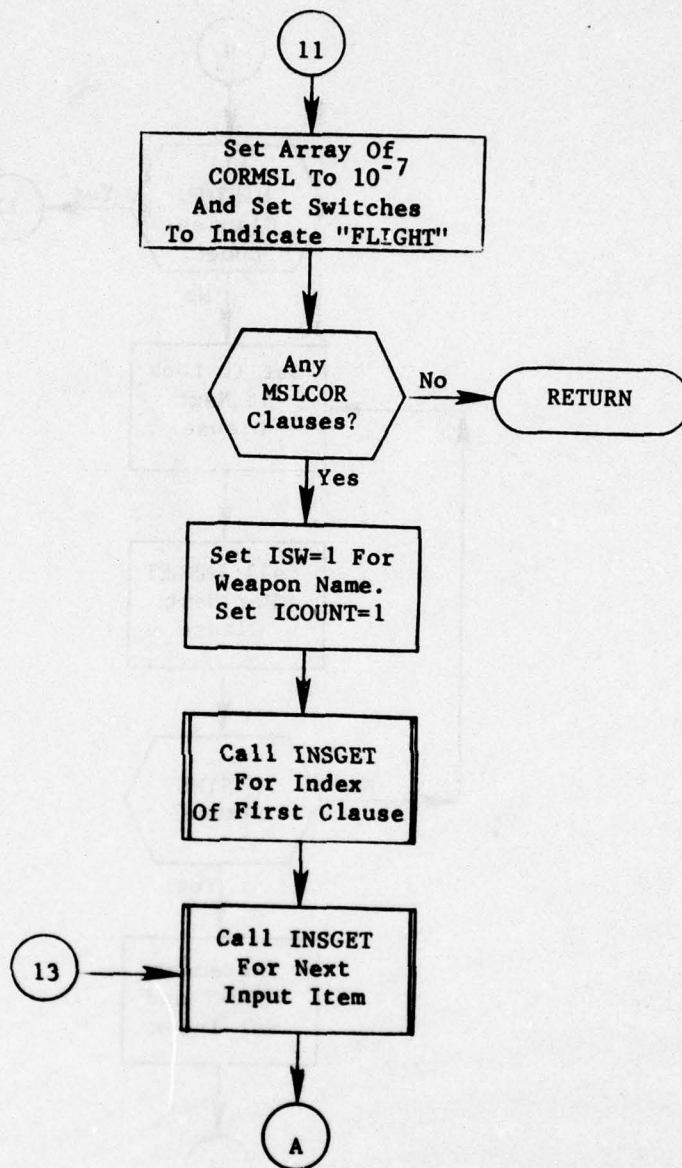


Figure 87. (Part 5 of 9)

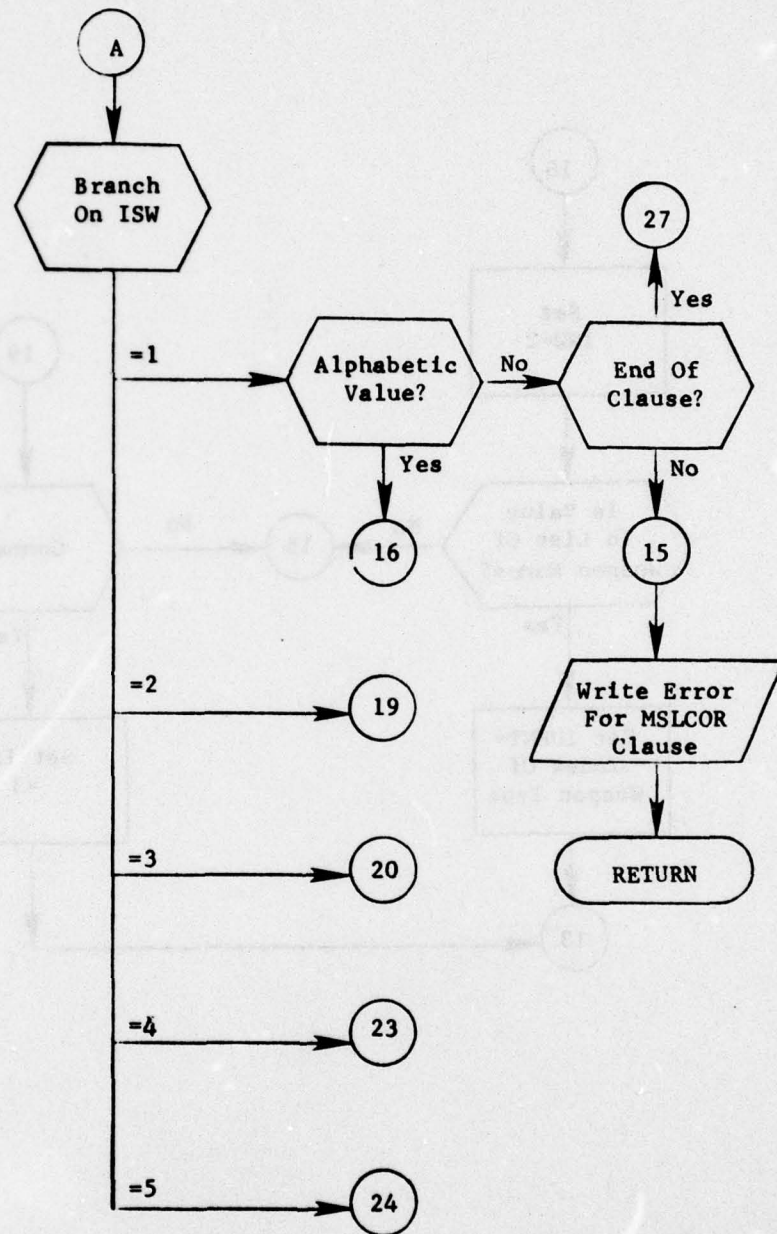


Figure 87. (Part 6 of 9)

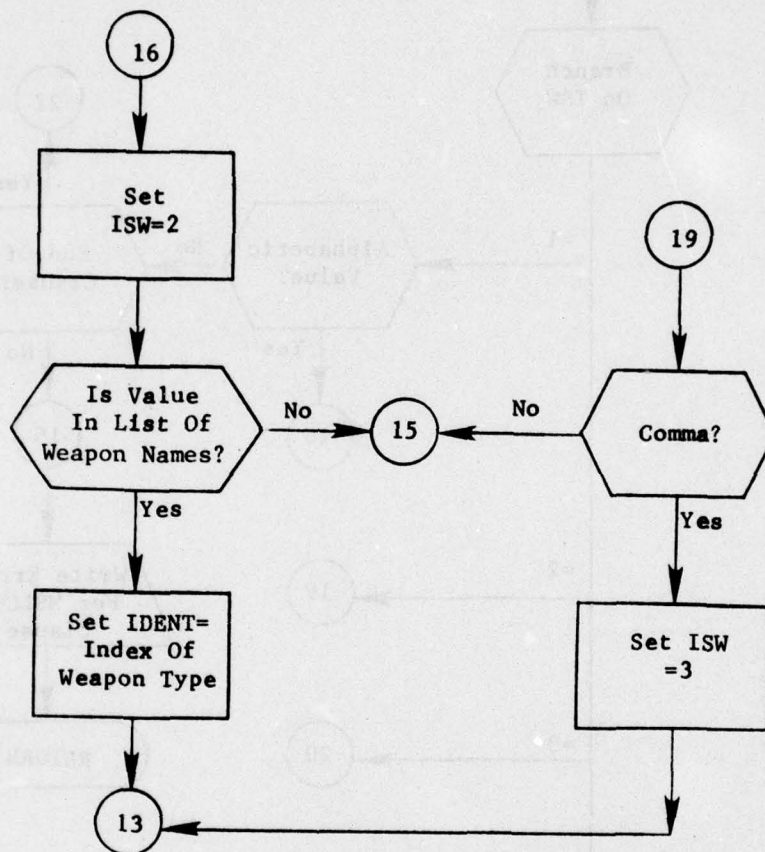


Figure 87. (Part 7 of 9)

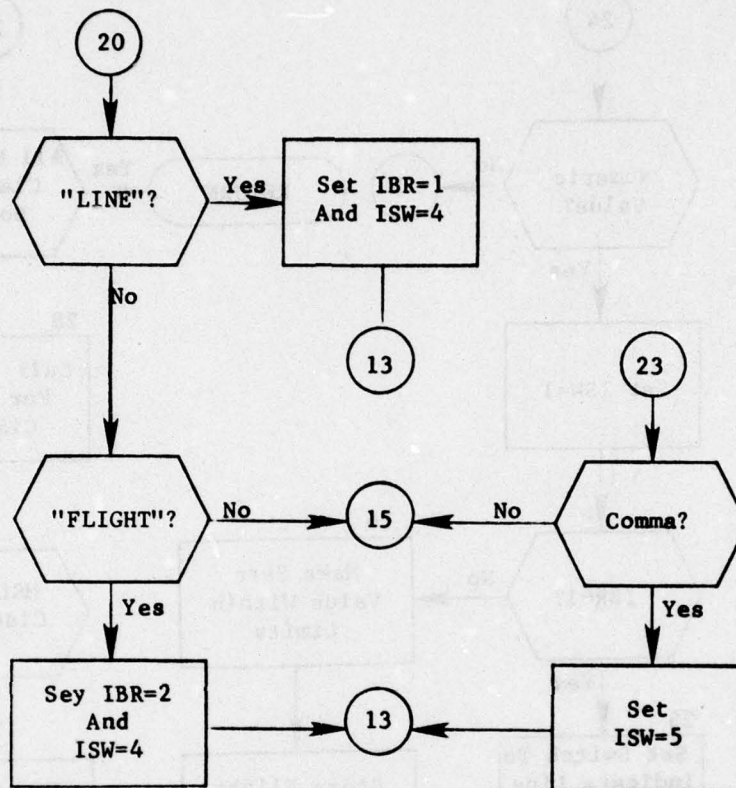


Figure 87. (Part 8 of 9)

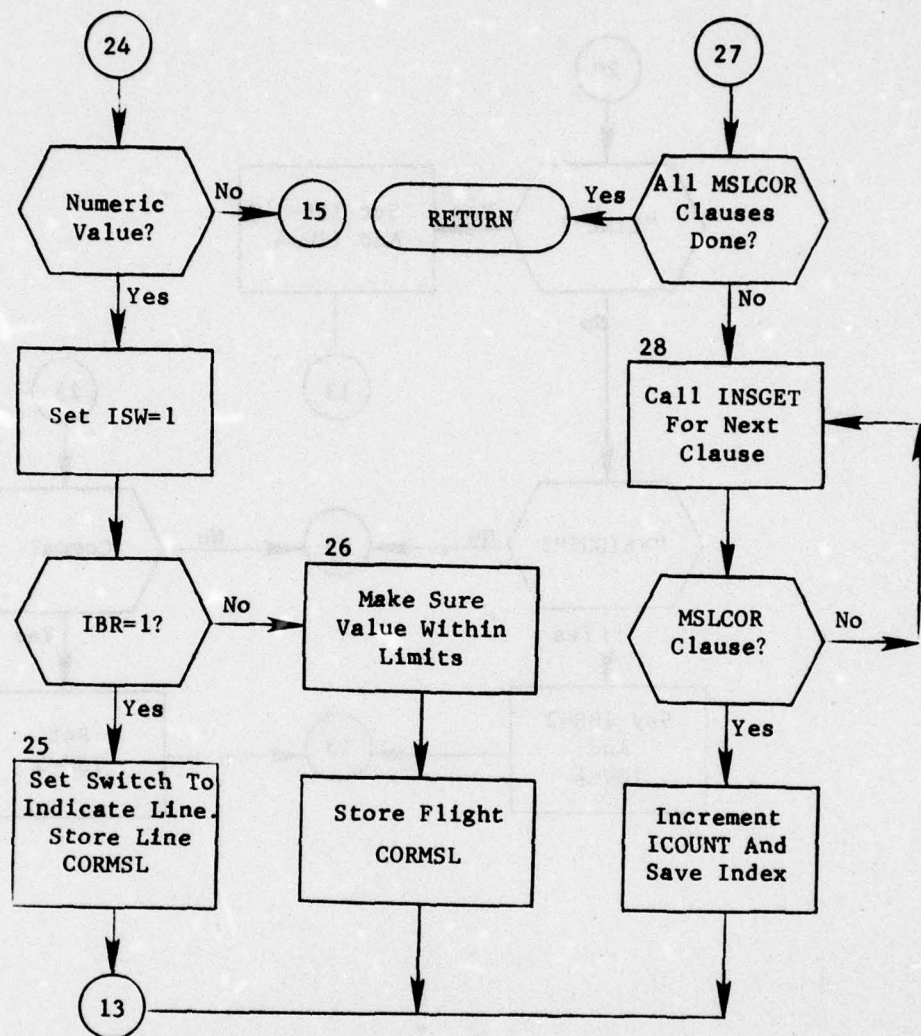


Figure 87. (Part 9 of 9)

4.8.13 Subroutine PLAN

PURPOSE: Subroutine PLAN develops detailed bomber sorties.

ENTRY POINTS: PLAN

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, ASMARRAY, C30, CONTROL, CONTR1, CORCOUNT, CORRCHAR, DECA, DINDATA, DISTC, DPENREF, EVENTS, GRPSTF, HAPPEN, HILO, ICLASS, IDP, IGO, IOUT, IRF, LASM, MH2, OUTSRA, OUTSRT, PAYSTF, POLITE, PPINFO, PPXX, RECBAS, RECOVERY, SPASM, TANKA, TEMPO, TYPSTF, VICINITY, WHDSTF

SUBROUTINES CALLED: ADJUST, CLINDATA, DISTF, GLOG, INTERP, IPUT, ITLE, LEREORDER, ORDER, POSTFLY, POSTLAUN, POST17, POST4, POST8, PREFL1, PREFL2, REORDER, SLOG, SNAPIT

CALLED BY: PLANBOMB

Method:

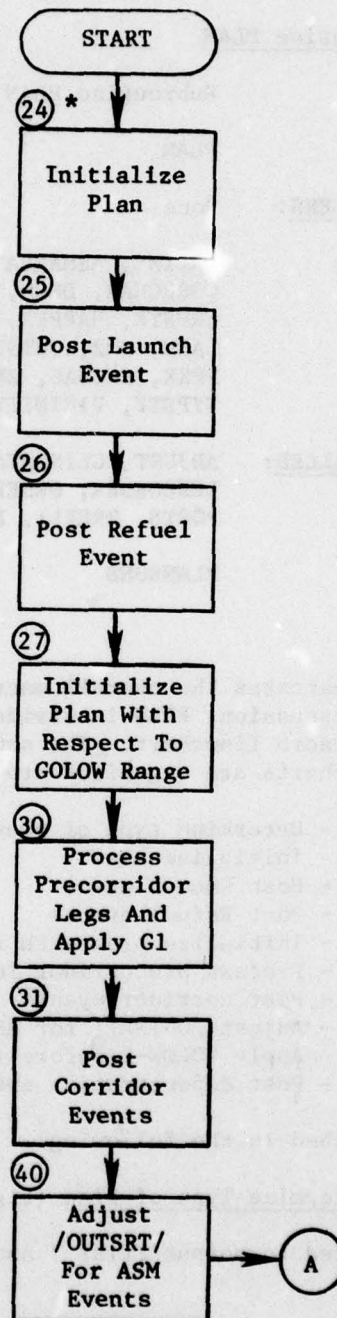
Figure 88 illustrates the overall macro flowchart for subroutine PLAN. To simplify discussion, PLAN is divided into "blocks" of coding as noted in the macro flowchart. The subroutine description as well as detailed flowcharts are organized around these blocks, which are:

- BLOCK 20 - Determine type of plan
- 24 - Initialize plan
- 25 - Post Launch event
- 26 - Post Refuel events
- 27 - Initialize plan with respect to GOLOW range
- 30 - Process precorridor legs and apply GOLOW-1
- 31 - Post corridor events
- 40 - Adjust /OUTSRT/ for ASM events
- 50 - Apply GOLOW-2 before first target
- 60 - Post depenetration events

and are described in the following.

Block 20: Determine Type of Plan (figure 89).

SNAPIT is called to output Print 7 and the low altitude variable G1 is set.



* Circled numbers refer to start of coding blocks rather than to statement numbers.

Figure 88. Subroutine PLAN (Macro Flowchart)
(Part 1 of 2)

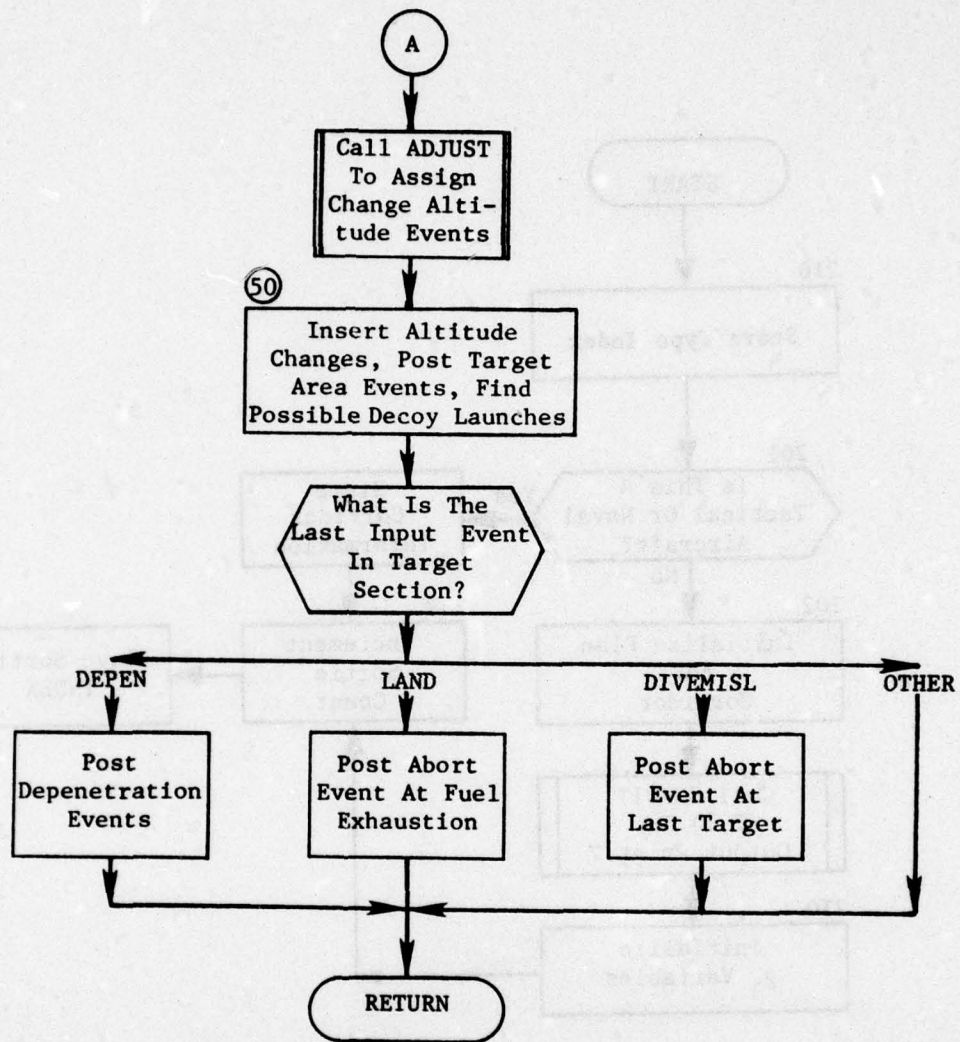


Figure 88. (Part 2 of 2)

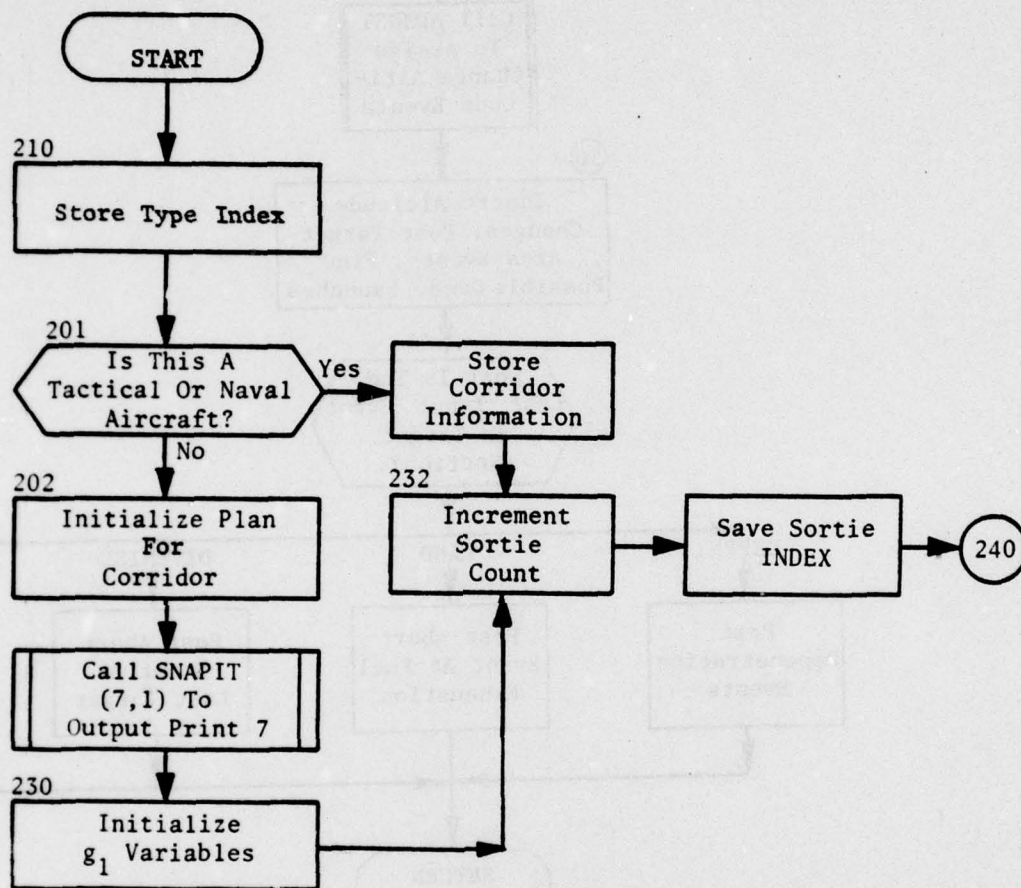


Figure 89. Subroutine PLAN
Block 20: Determine Type of Plan

SNAPIT is called to output Print 7 and the low altitude variable G1 is set.

Block 24: Initialize Plan (figure 90)

The plan initialization at block 24 consists of setting pointers and indices for the appropriate depenetration corridor and writing other information into the final plan. Indices for the Payload table and ASM table are set at this point, as are parameters which are dependent on bomber speed. These parameters are associated with the minimum length of time a bomber flies low, and where a bomber is to change altitude in the neighborhood of a target.

Block 25: Post Launch Event (figure 91)

After initialization, the posting of events in the output arrays of common /DINDATA/ begins with the posting of the Launch event. It is posted by location (latitude and longitude) as well as by type and "place" index. Table 10 lists the types of events admissible for posting, as well as their names. Note that the GO HIGH, GO LOW, and DOGLEG events are not admissible in the final plan.

Block 26: Post Refuel Events (figure 92)

If there is a Refuel event, it is posted next. Refueling is accomplished in one of three ways: (1) at preassigned refuel areas (refuel index ≥ 0), (2) buddy refueling by another bomber or tanker launched from the same base (refuel index = -1 or -2), or (3) automatically by PLAN (refuel index = -4 for single refuel, -5 for two refuels). In the first case the data preparer assigns refuel areas for both bombers and tankers. In buddy refueling, tankers are ignored. Bombers are refueled by the buddy system at maximum range (a great-circle distance from the base equal to refueled range minus range) or just prior to the corridor origin, whichever is sooner.

When a bomber is to be assigned a refuel area by PLNTPLAN the buddy refuel point, X, is first computed a distance ΔR from the base on a great circle between it and the corridor entry, as for buddy refueling. See figure 93. ΔR is the difference between refueled range and range. If there already exist refuel areas which are within ΔR of the base and within some specified distance, D, of the point X, the area nearest X is assigned as the refuel point. Otherwise the point X is assigned and is added to the list of refuel areas.

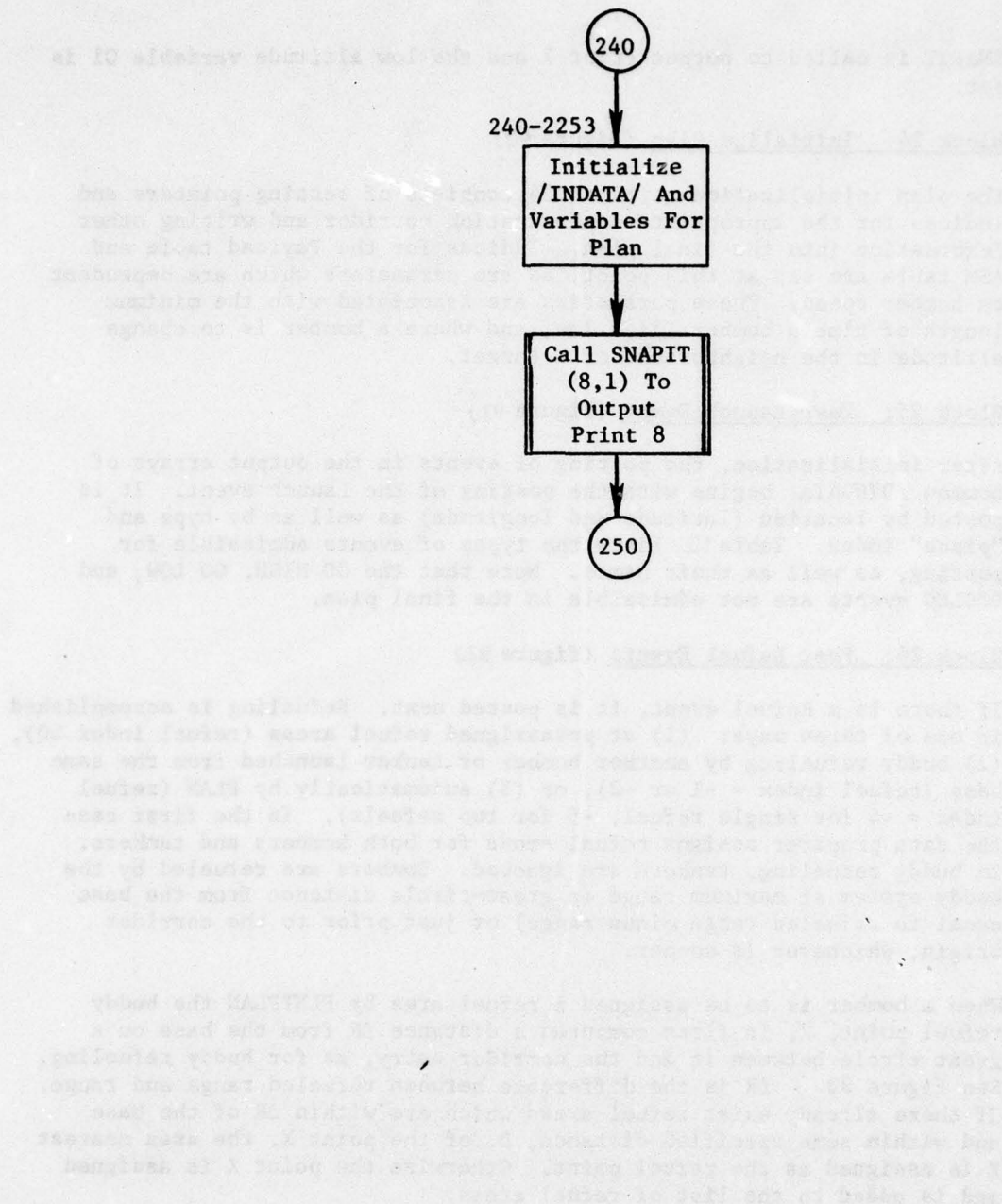


Figure 90. Subroutine PLAN
Block 24: Initialize Plan

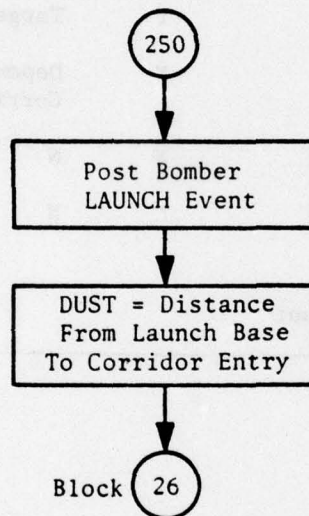


Figure 91. Subroutine PLAN
Block 25: Post Launch Event

Table 10. List of Admissible Input Events by Type and Information Relevant to Each

<u>TYPE OF HAPPENING</u>	<u>(EVENT CODE)</u>	<u>LAT., LONG</u>	<u>PLACE</u>	<u>WEAPON OFFSET LAT., LONG.</u>
DOGLEG	(20)	Y	N	N
DROPBOMB	(8)	Y	Target Index	Y
AIM ASM	(14)	Y	Target Index	Y
DEPEN	(16)	N	Depenetration Corridor Index	N
LAND	(13)	N	N	N
DIVEMISL	(16)	N	N	N
Y = relevent				
N = not relevant				

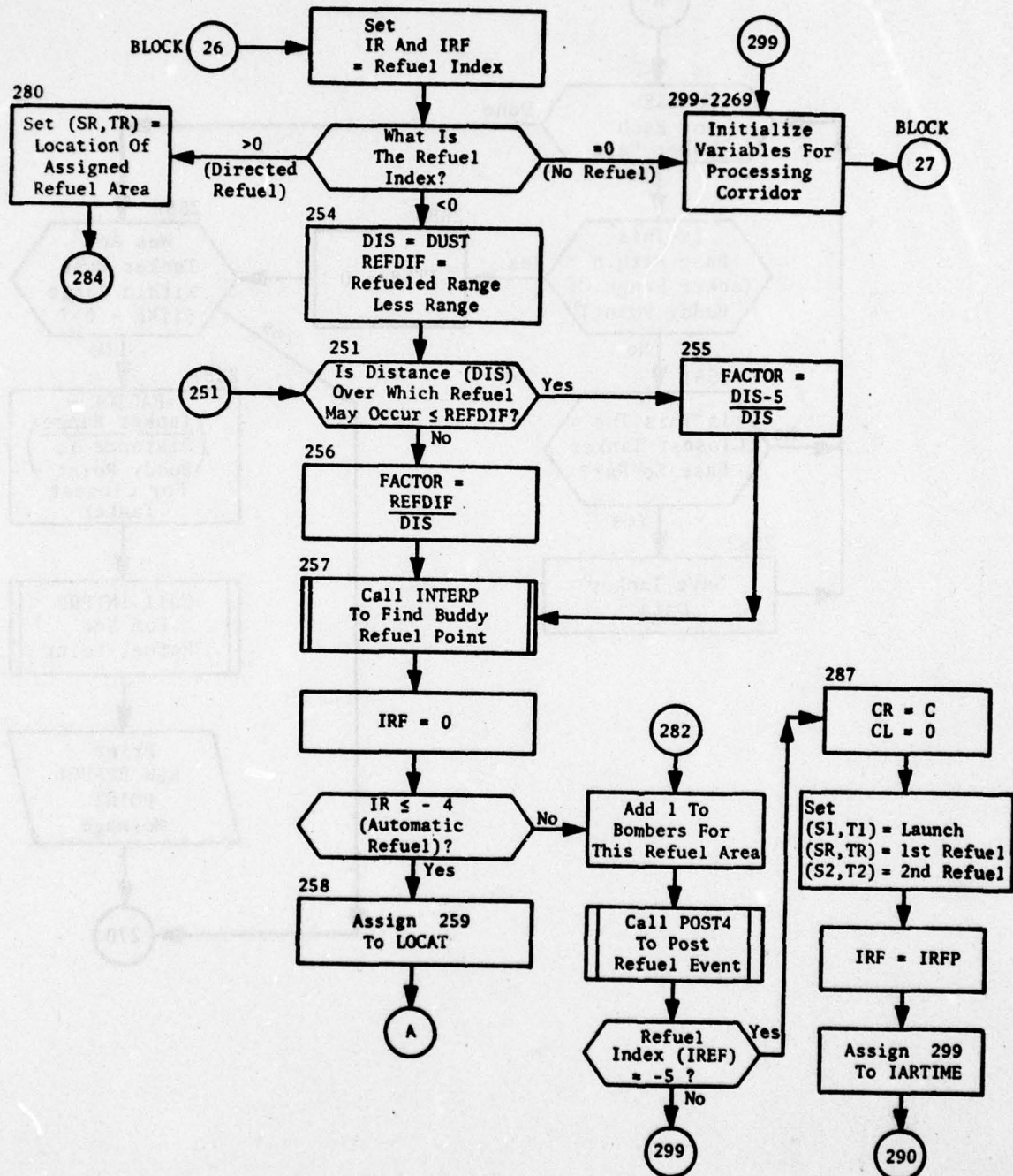


Figure 92. Subroutine PLAN
Block 26. Post Refuel Events
(Part 1 of 5)

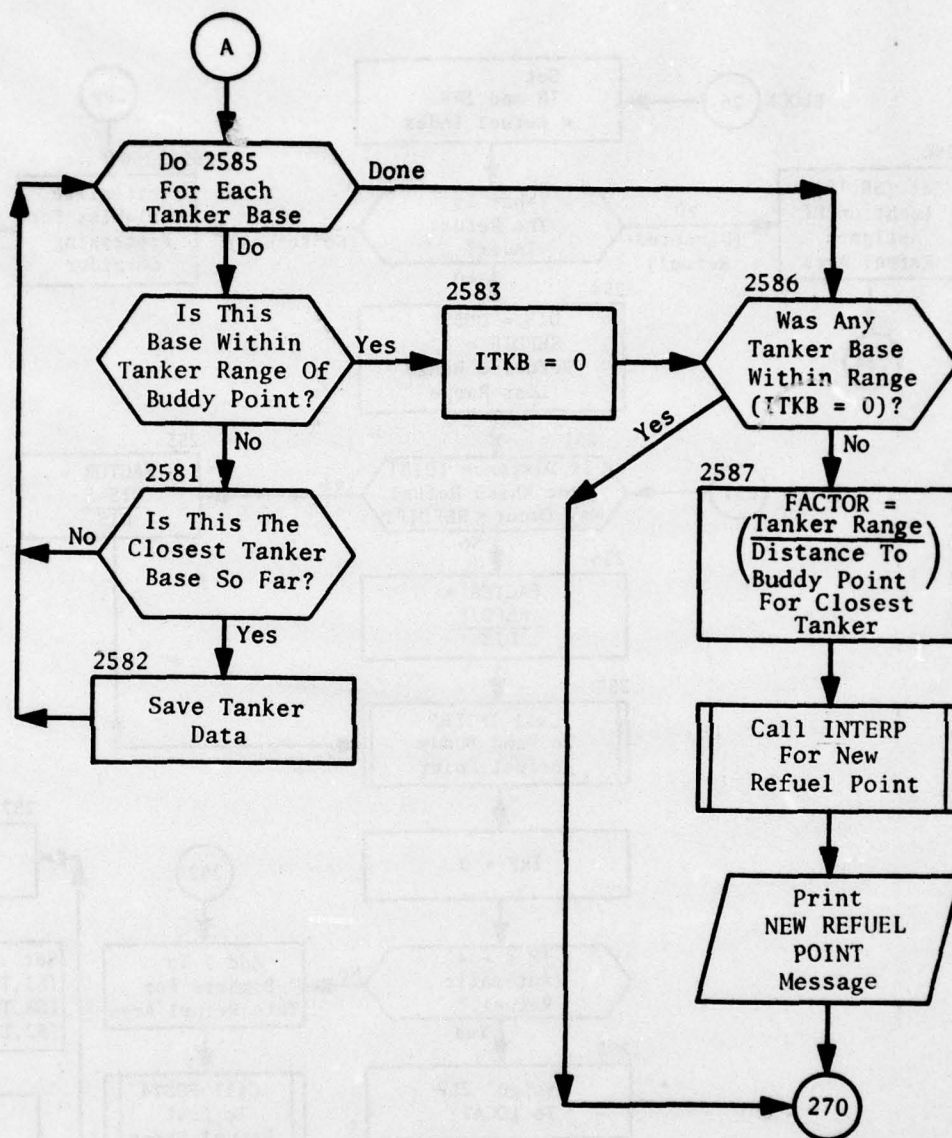


Figure 92. (Part 2 of 5)

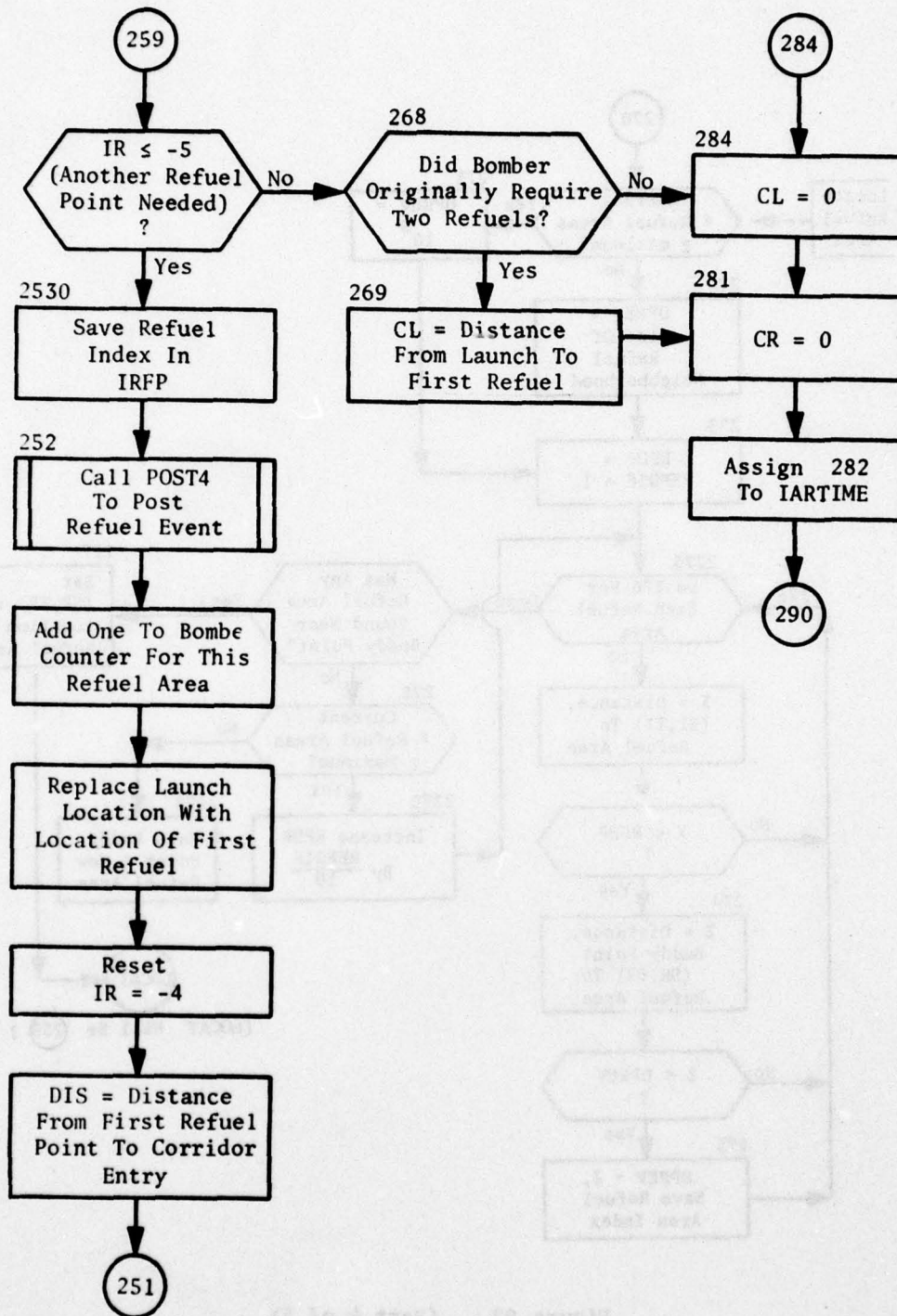


Figure 92. (Part 3 of 5)

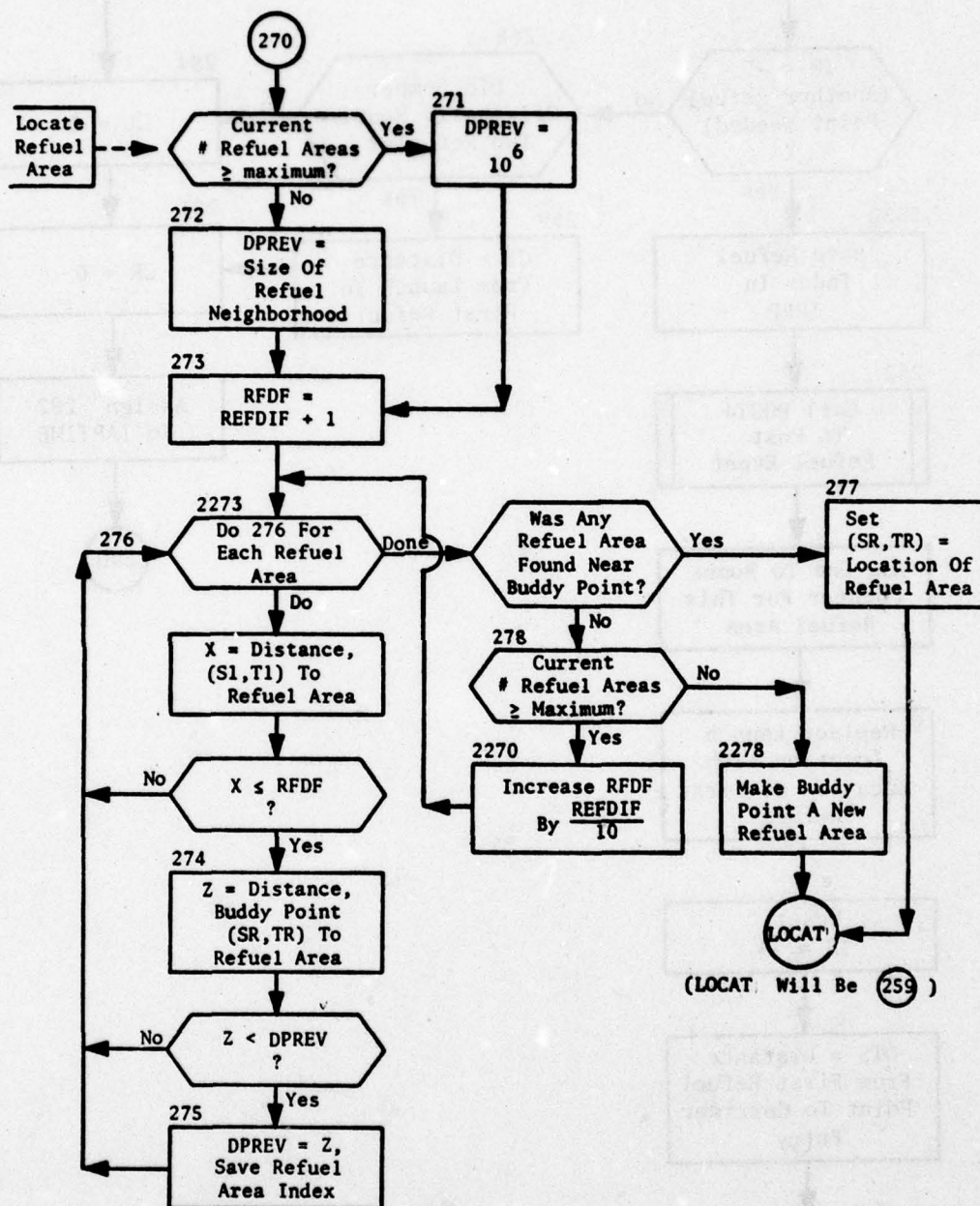
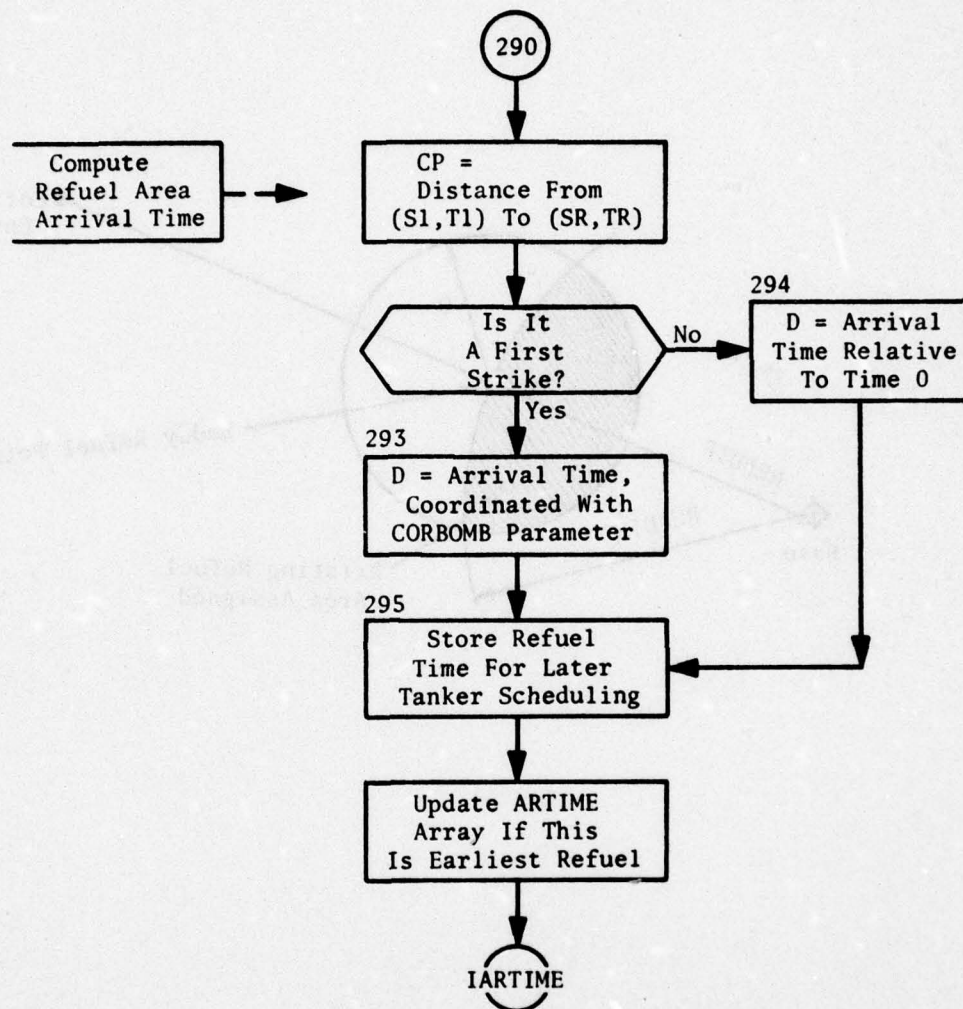


Figure 92. (Part 4 of 5)



(IARTIME May Be 282 or 299)

Figure 92. (Part 5 of 5)

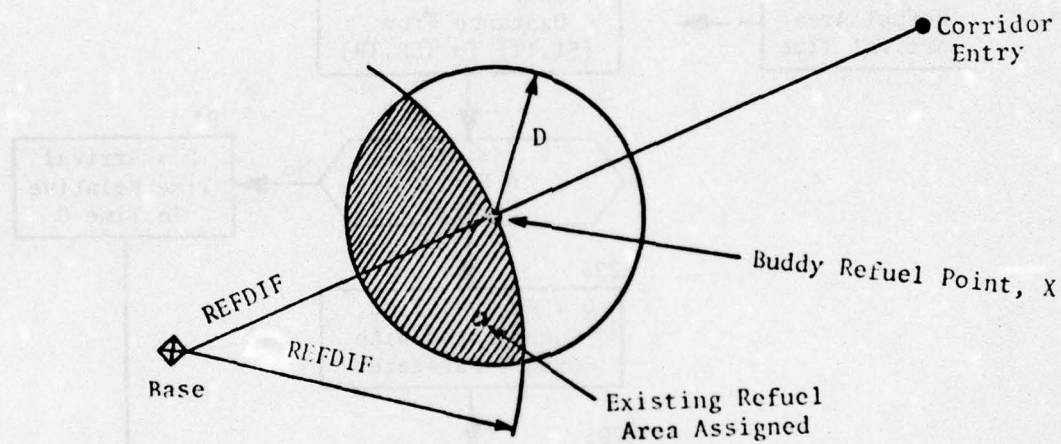


Figure 93. Acceptable Locations for Refuel Area (Shaded Section)

The list of tanker bases is then scanned to see whether the buddy refuel point is within range of any of them. If not, the closest tanker base is chosen and a new refuel point is computed by interpolation. This point will lie on the line drawn between the tanker base and the original refuel point in such a way that it will be within range of the tanker base.

The actual time of arrival at the refuel area is computed, using the CORBOMB parameter if the plan is for a first strike. The earliest arrival time in each refuel area is saved for later use when generating tanker plans (array ARTIME). Also saved for tanker scheduling is the arrival time and refuel area for each bomber (array ARVLS).

Block 27: Initialize Plan with Respect to GOLOW Range (figure 94)

The low-altitude range available to the bomber in flying the sortie is specified to PLAN in three separate amounts: the amount during the precorridor legs (G_1), the amount immediately prior to the first target (G_2), and finally, the amount immediately following the first target (G_3).

In block 27, these amounts are examined to make certain that the bomber does not fly low for less than 15 minutes. If $G_1 < 15 * SPDHI$, then G_1 is added to G_2 . If $G_2 + G_3 < 15 * SPDHI$, then G_2 and G_3 are set to zero.

If the bomber is a tactical or naval aircraft (denoted by the use of corridor 1 or 2), coding blocks 30 and 31 are skipped.

Block 30: Process Precorridor Legs and Apply GOLOW-1 (figure 95)

The main sortie processing begins then at block 30 with the processing of the precorridor legs. They must be processed in the opposite direction from the bomber flight beginning at the origin and proceeding backward toward the entry. This is because the available low-altitude range (G_1) is measured backward from the corridor origin. Corridor attrition may be associated with the precorridor legs, and low-altitude range is applied against only those corridor sections where the bomber would experience attrition. Any G_1 remaining is added to G_2 .

The processing for this block of coding is perhaps best described by referring first to figure 96 which gives an example of precorridor legs in the most complex configuration allowed. It also shows how this corridor is described to the module in /HAPPEN/. The corridor consists of eight separate doglegs or nine points, and so is described in nine lines in /HAPPEN/. Those doglegs where the bomber would experience listing the location (latitude and longitude) of each dogleg point in order beginning at the corridor origin and proceeding backward toward the corridor entry, as shown in the figure. With each point the distance from the previous point is also noted. If attrition begins at a point, this is noted by entering a 1, 2, or 3 in array JAPTYPE, depending upon whether this is the first, second,

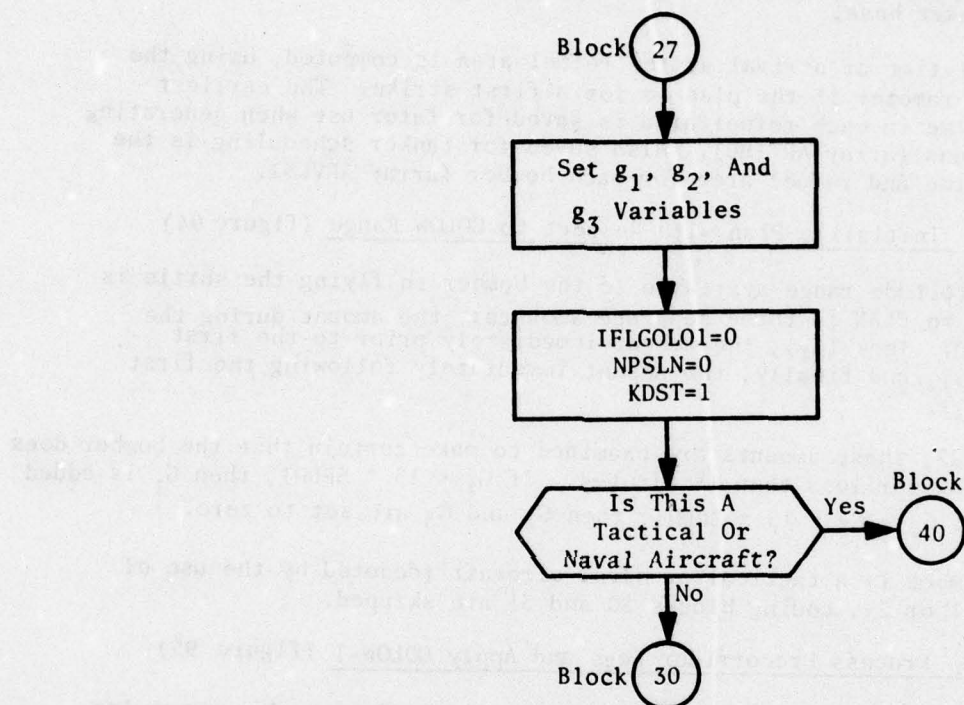


Figure 94. Subroutine PLAN
Block 27: Initialize Plan With
Respect to GLOW Range

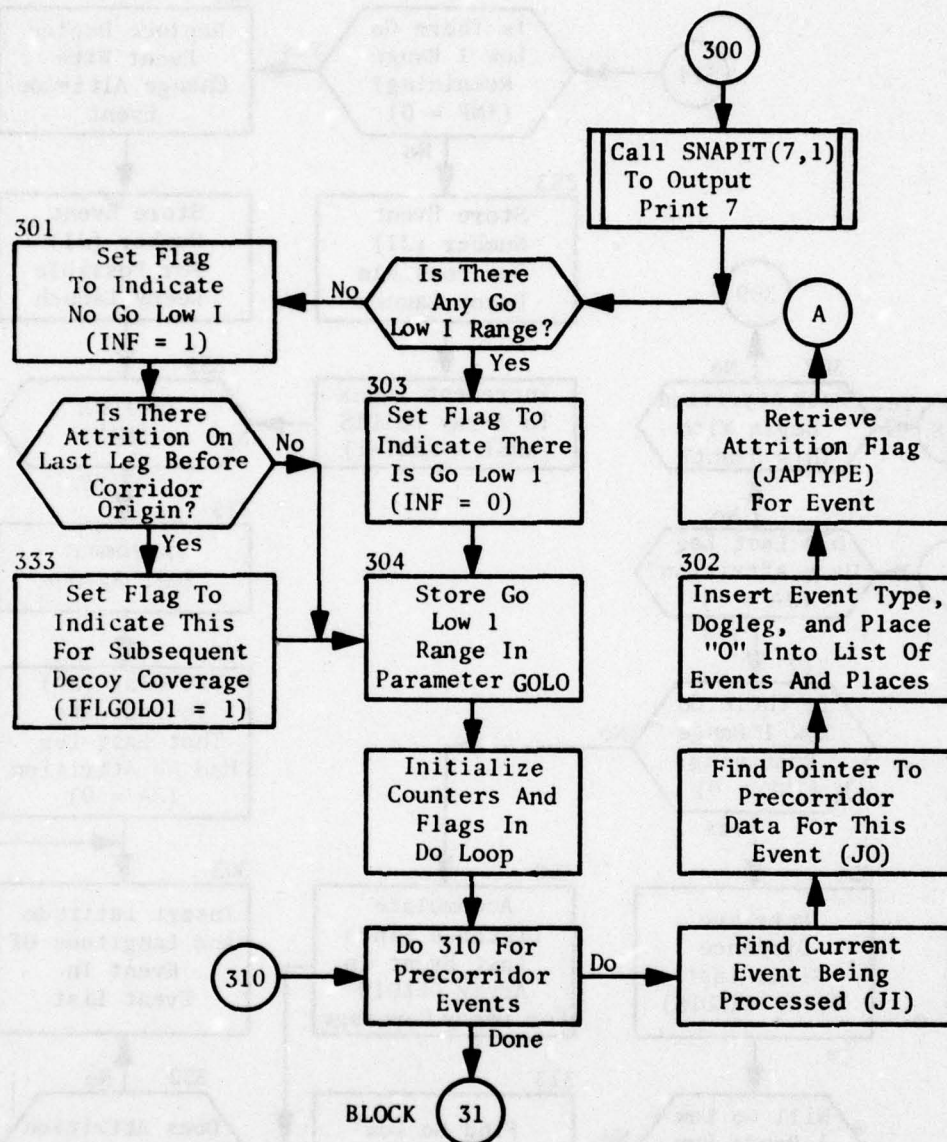


Figure 95. Subroutine PLAN
Block 30: Process Precorridor Legs and Apply GOLOW1
(Part 1 of 3)

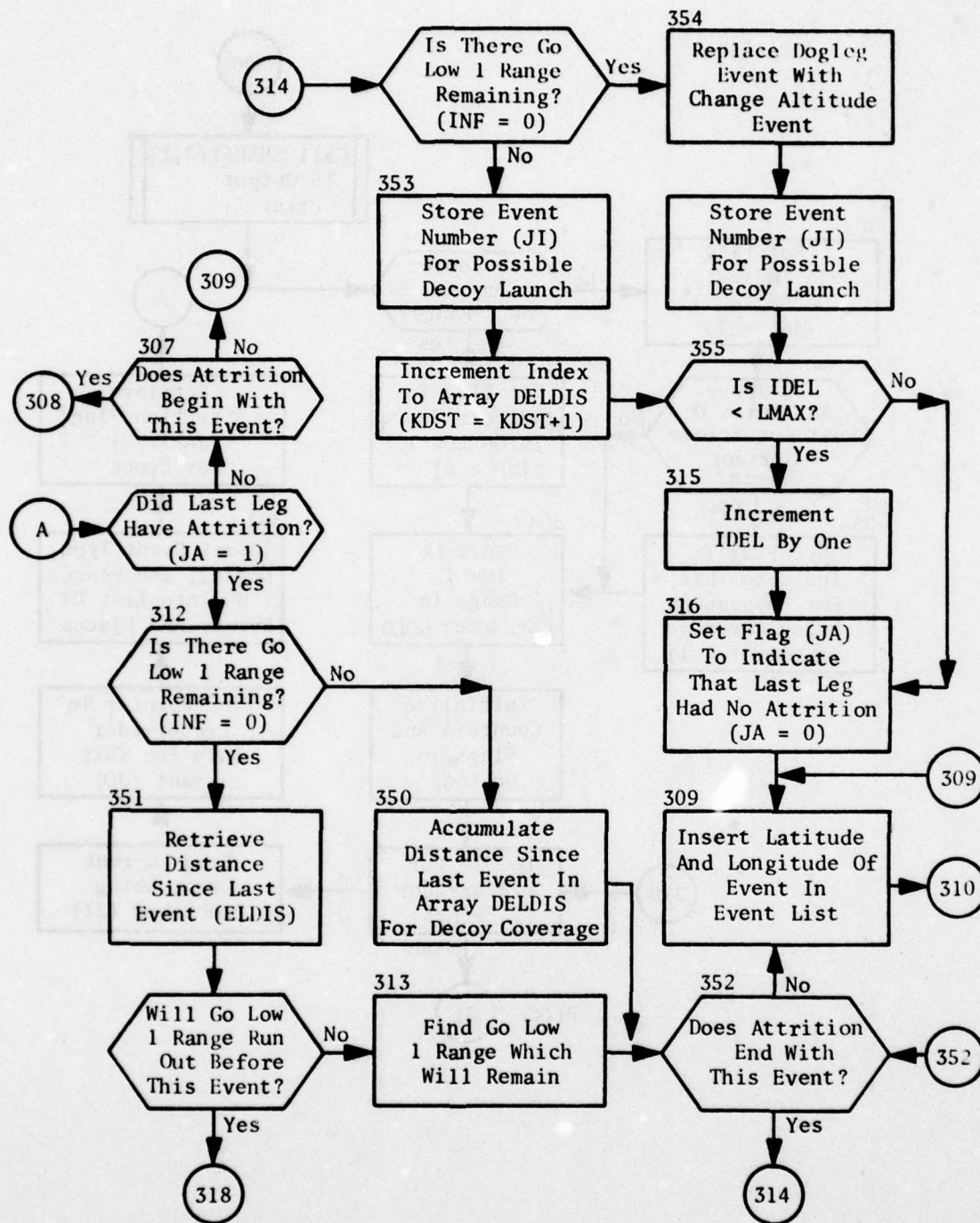


Figure 95. (Part 2 of 3)

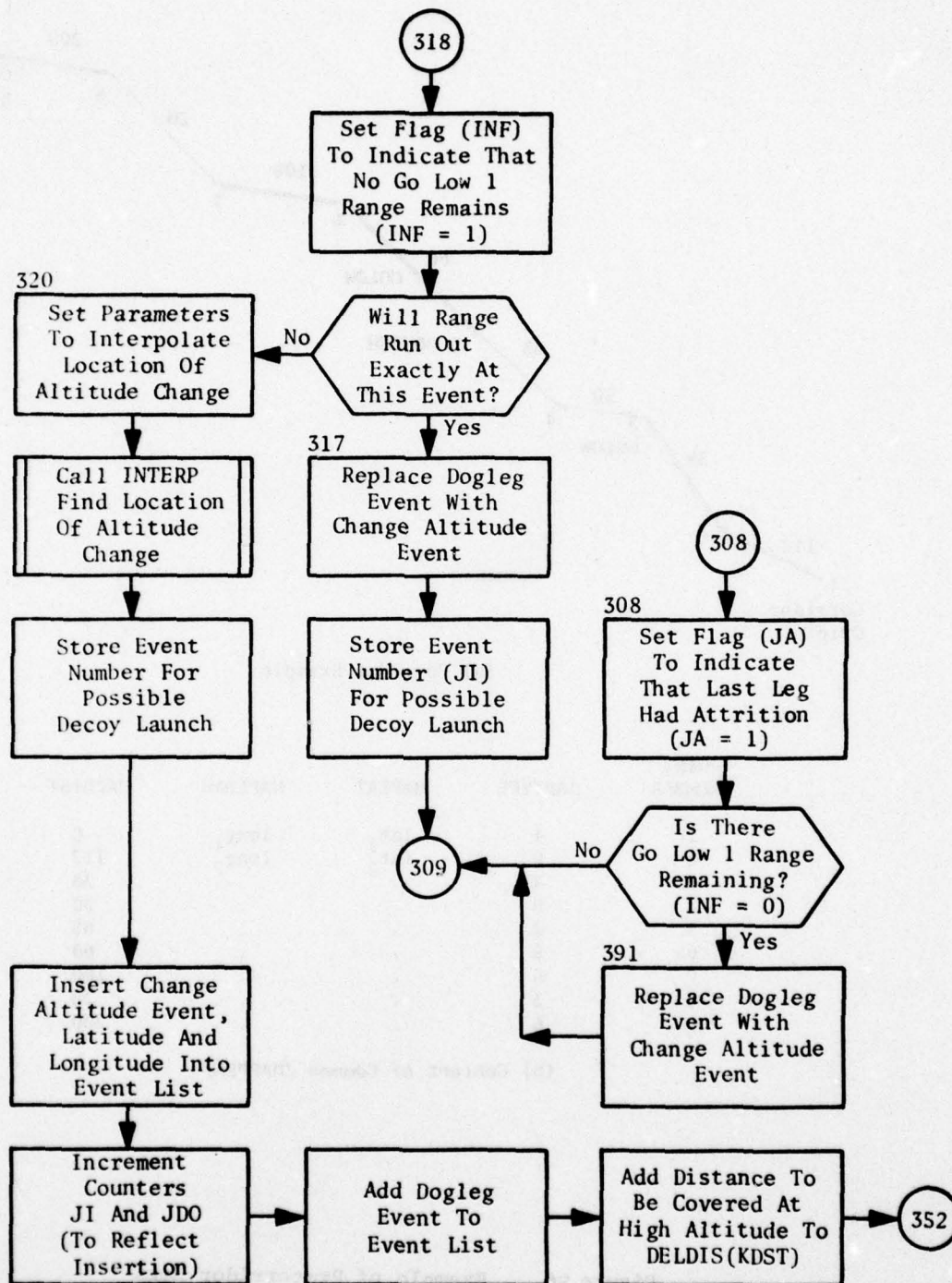
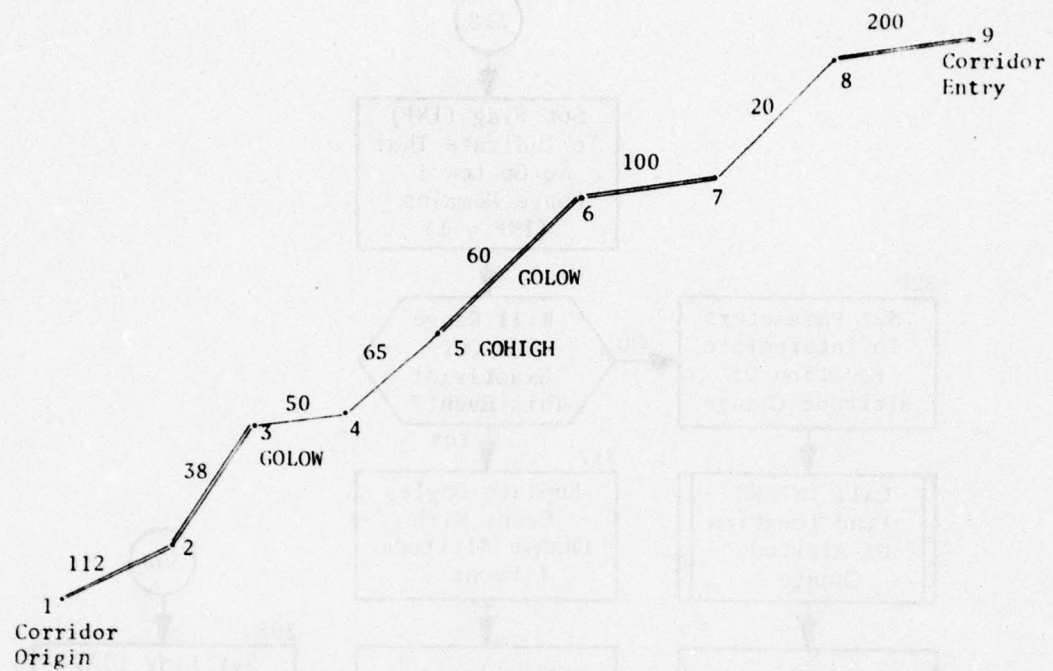


Figure 95. (Part 3 of 3)



(a) Corridor Example

POINT NUMBER	JAPTYPE	HAPLAT	HAPLONG	HAPDIST
1	1	lat ₁	long ₁	0
2	0	lat ₂	long ₂	112
3	4	.	.	38
4	0	.	.	50
5	2	.	.	65
6	5	.	.	60
7	0	.	.	100
8	3	.	.	20
9	6	.	.	200

(b) Content of Common /HAPPEN/

Figure 96. Example of Precorridor Legs

or third section. Similarly, if attrition ends at the point, the number 4, 5, or 6 is entered. Thus point 1 is labeled with a 1, and point 3 with a 4, to indicate the beginning and end of the first attrition section. This example, of course, describes an extreme situation where attrition occurs in three separate sections. Usually, there will be attrition in at most one section. The program must know which doglegs have attrition in order to know where to apply the low altitude range G_1 . In the figure example, suppose $G_1 = 180$ miles, then 112 miles would be applied against the first dogleg back of the corridor origin, 38 miles applied to the second dogleg. The balance of 30 miles would be applied to the 5th dogleg beginning with point 5 and ending midway between points 5 and 6. As a result, GO LOW and GO HIGH events would be posted as indicated in the figure. The posting of a GO HIGH at the corridor origin depends on the value of G_2 .

This section also sets up arrays which contain the event numbers of all those precorridor events which might possibly call for the launching of a decoy. These arrays are called LOHIMHT and LDMHT. The event number is stored in LOHIMHT for events of priority 2. (See subroutine DECOYADD for table of priorities.) The event number is stored in LDMHT for a launch of priority 3 (or, after the first such event, priority 5). For the priority 5 launches, the distance to be covered by the decoy is accumulated in a corresponding word of array DELDIS.

The flag JA is set to 1 when the beginning of an attrition section is encountered, and back to 0 at the end. IDEL is the indicator to be compared against JAPTYPE in order to determine the beginning and end of attrition events. The counter JDO is advanced each time an additional Change Altitude event is added.

Block 31: Post Corridor Events (figure 97)

G_1 is measured out and the necessary change altitude events determined. The possible Decoy Launches then are posted by filling array LMHT with the event number (from array LDMHT or LOHIMHT) and by filling array LPRIORITY with the associated priority (5 if the event number is from array LDMHT, 2 if the event number is from array LOHIMHT). For the priority 5 launch, the index to the associated distance in array DELDIS is the same as the index to the event number in array LDMHT; hence, this index is stored in array NDCYRQ. The priority 2 launch does not require a coverage distance. Whenever this is the case, a 1 is stored in array NDCYRQ. The actual filling of these arrays is done by subroutine POSTLAUN.

Block 40: Adjust /OUTSRT/ for ASM Events (figure 98)

The list of input events in the /OUTSRT/ arrays is next examined for ASM events. If there are any, the aim or launch points for them are now calculated. If necessary, the ASM events are reordered within the list of other events at this time. This is because ASM target events are

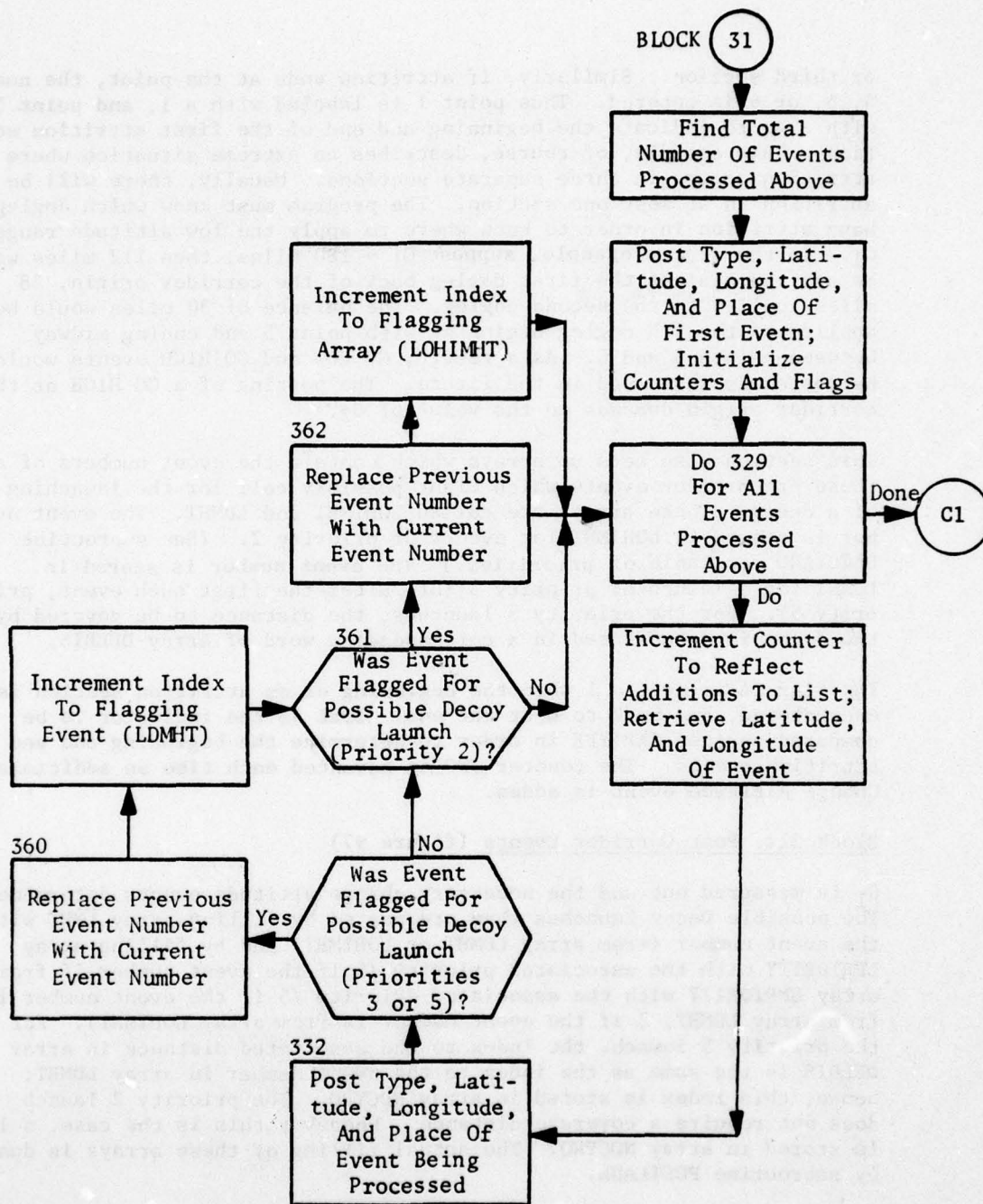


Figure 97. Subroutine PLAN
Block 31: Post Corridor Events
(Part 1 of 4)

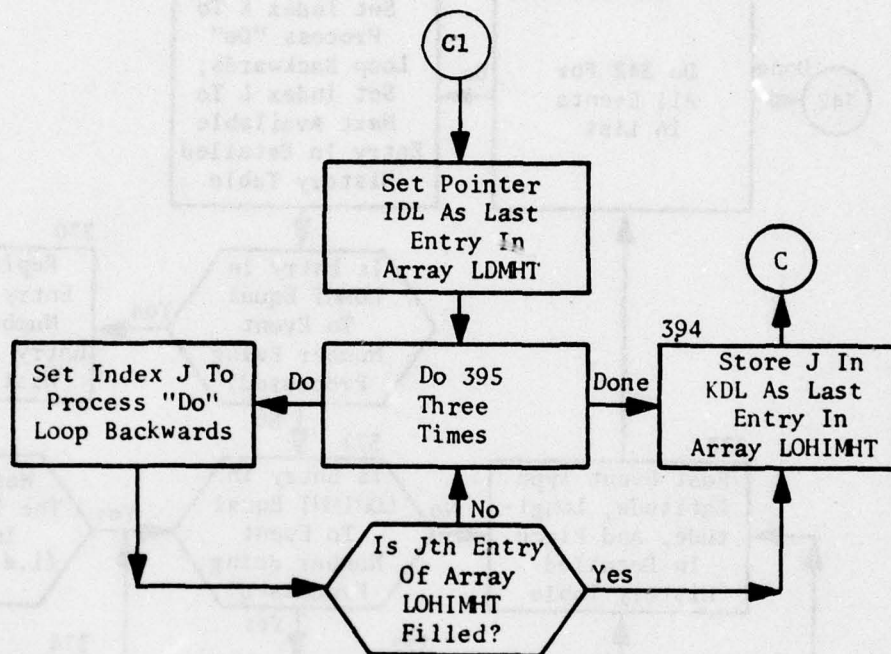


Figure 97. (Part 2 of 4)

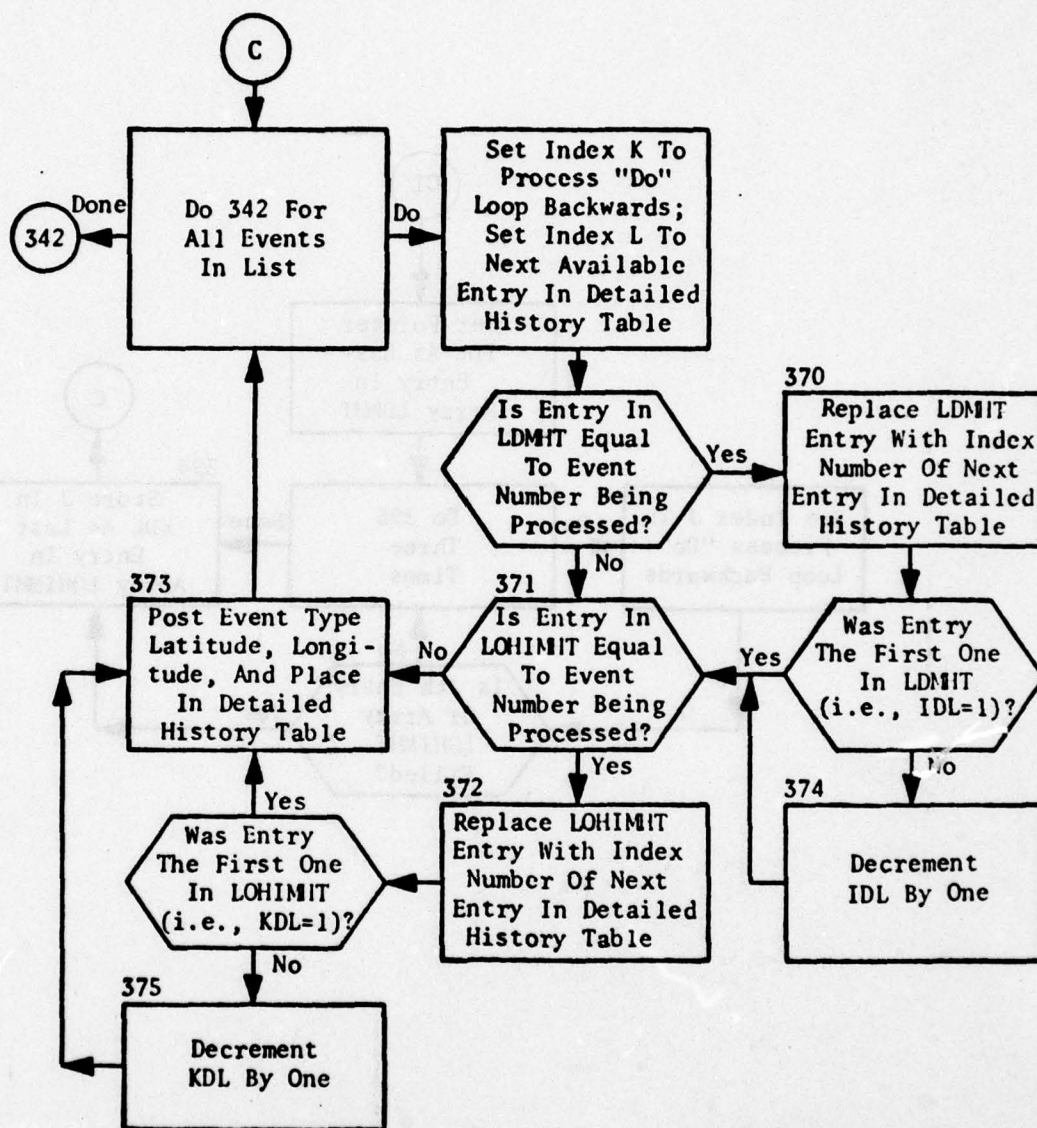


Figure 97. (Part 3 of 4)

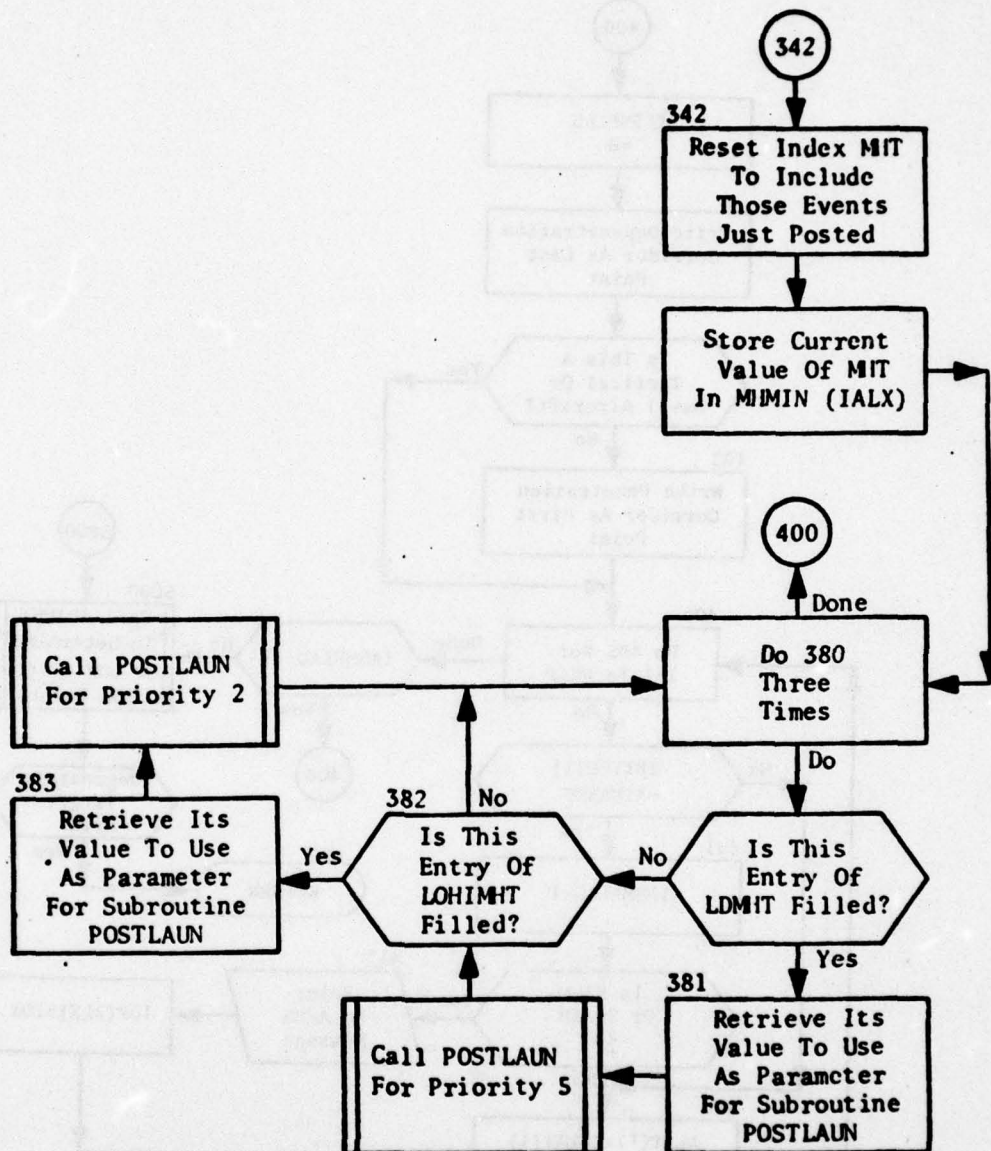


Figure 97. (Part 4 of 4)

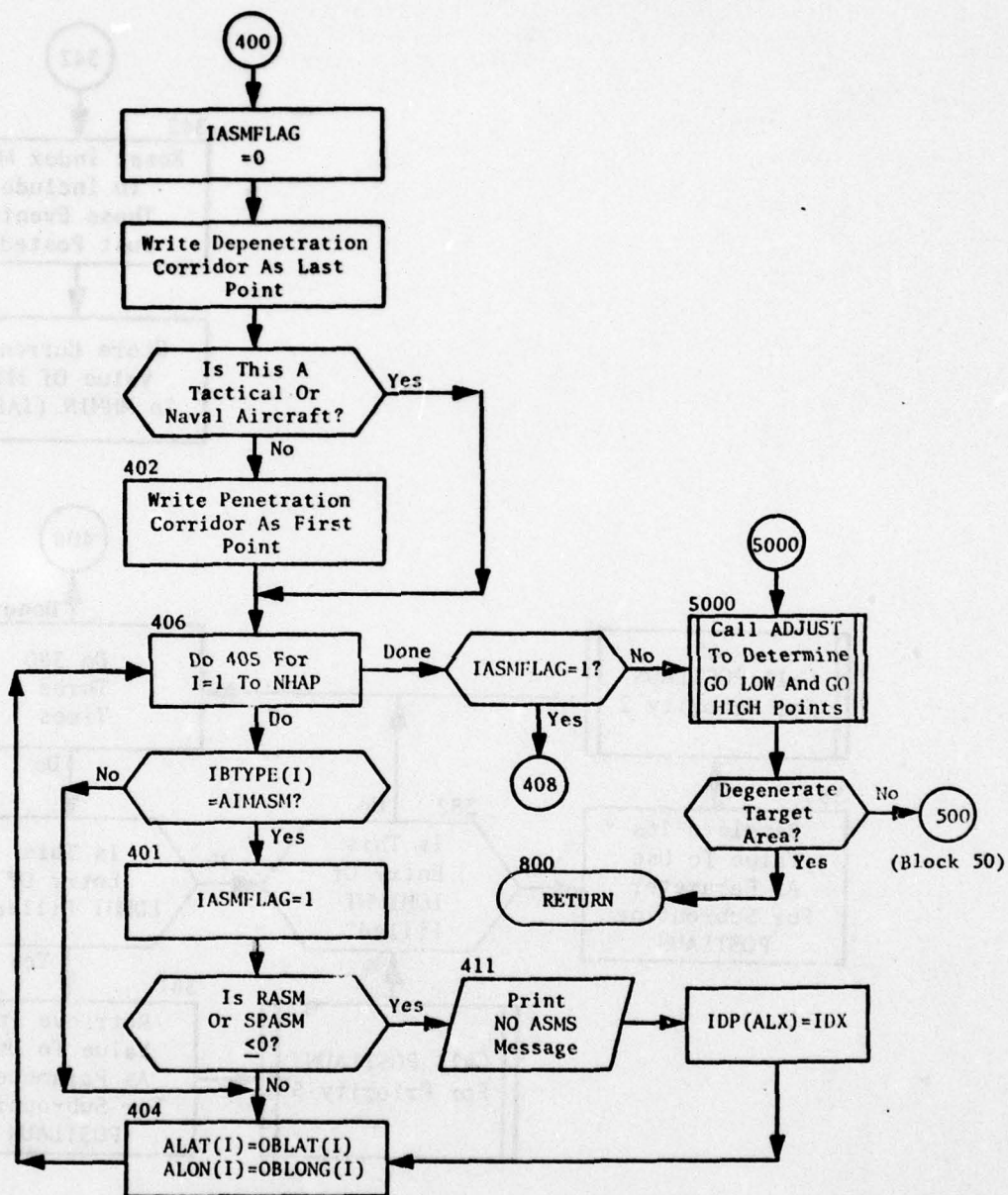


Figure 98. Subroutine PLAN (Part 1 of 4)
Block 40: Adjust /OUTSRT/ for ASM Events

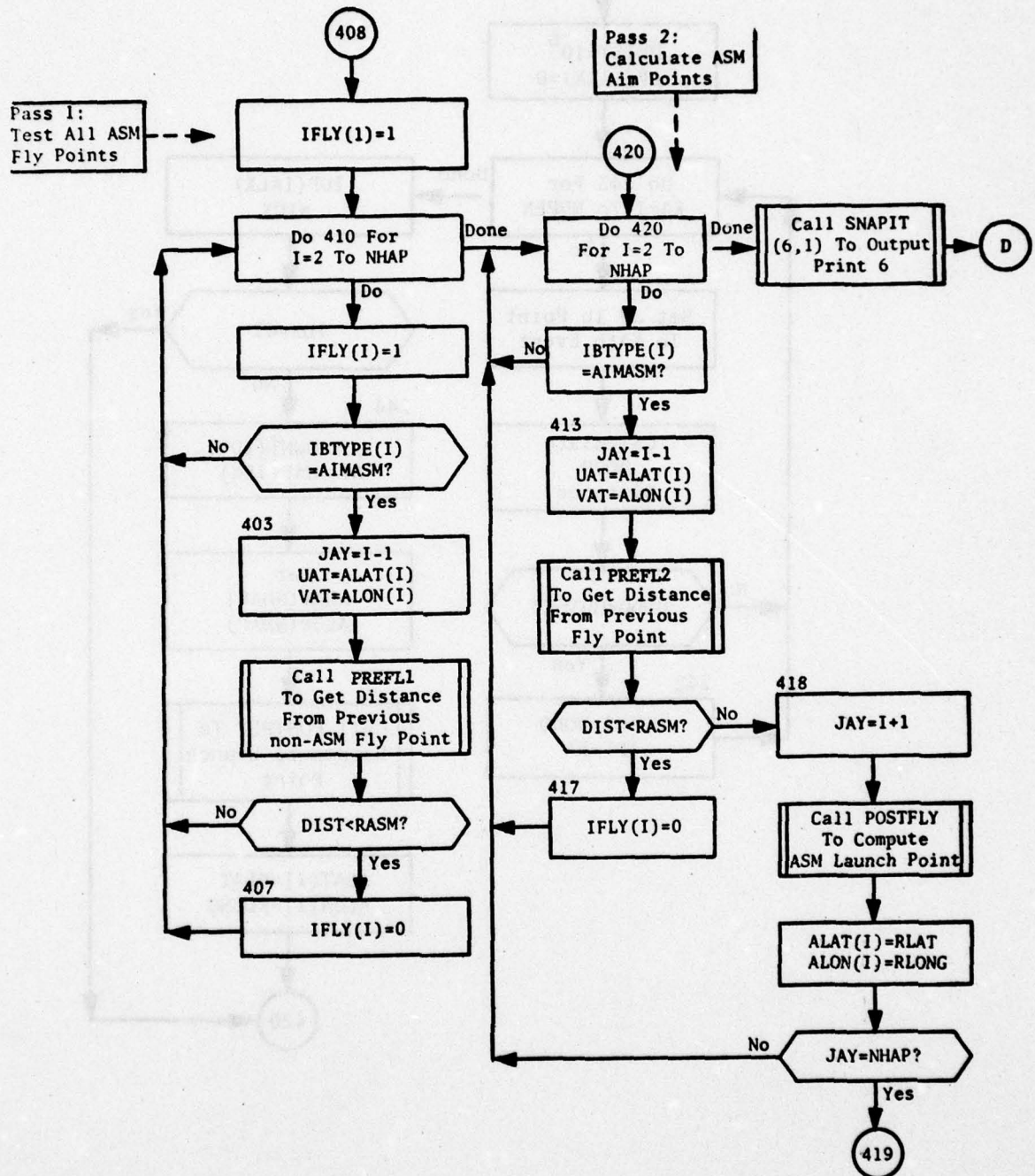


Figure 98. (Part 2 of 4)

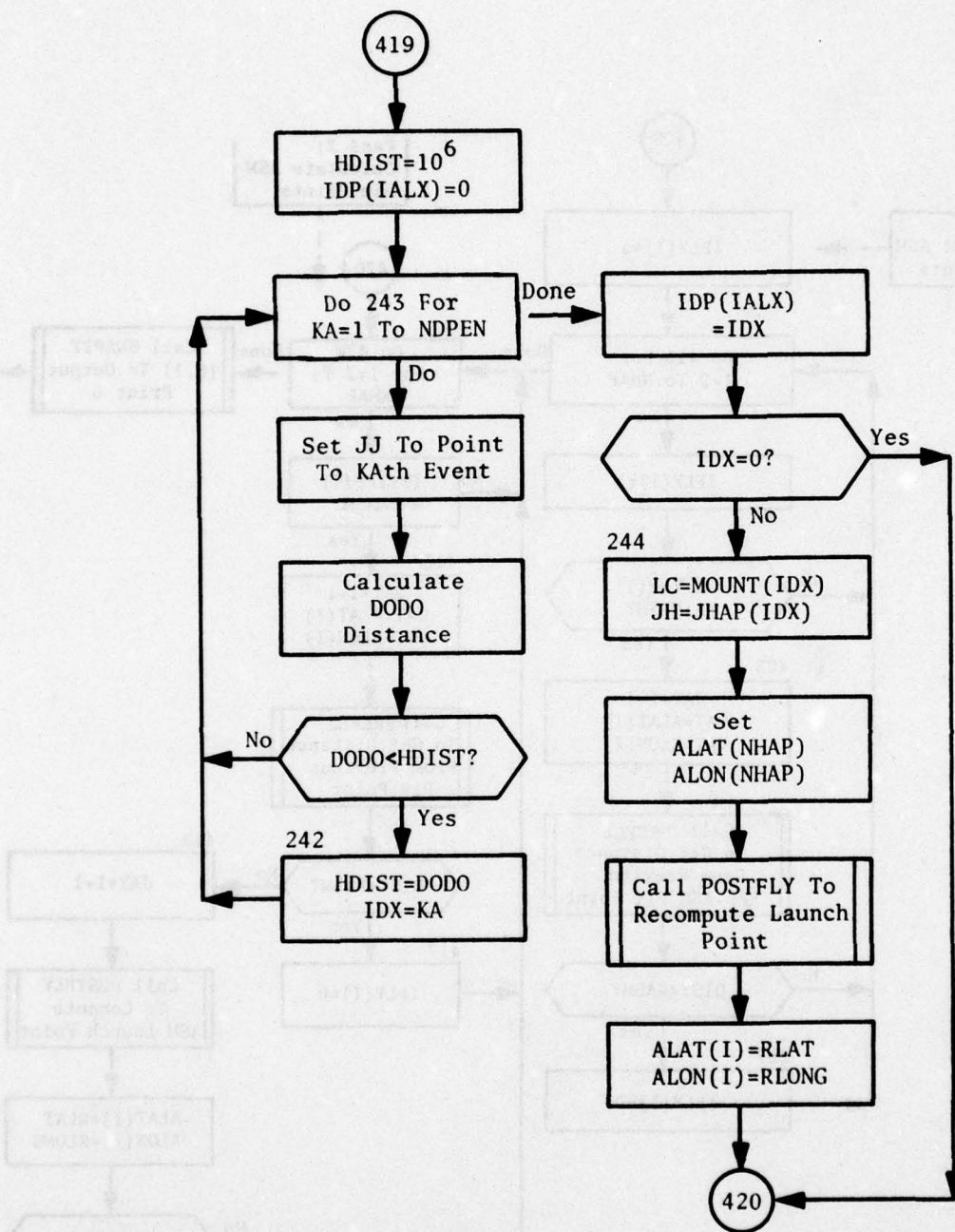


Figure 98. (Part 3 of 4)

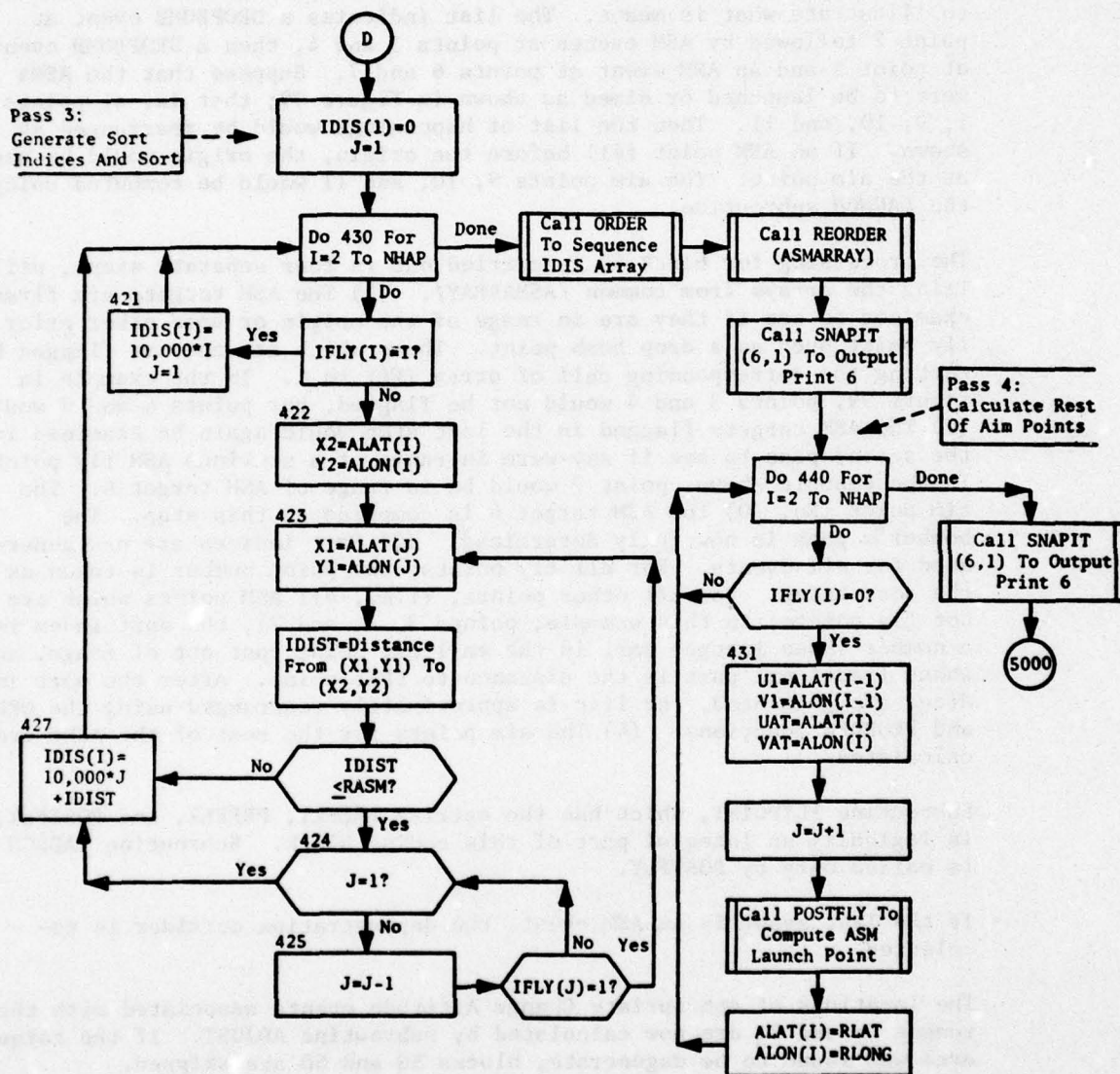


Figure 98. (Part 4 of 4)

supplied to PLANOUT by POSTALOC without aim points, and approximately in their proper order. They may appear later in the lists than they should, but not earlier. Figure 99 shows a list of happenings with ASM events, to illustrate what is meant. The list indicates a DROPBOMB event at point 2 followed by ASM events at points 3 and 4, then a DROPBOMB event at point 5 and an ASM event at points 6 and 7. Suppose that the ASMs were to be launched or aimed as shown in figure 99; that is, at points 1, 9, 10, and 11. Then the list of happenings would be rearranged as shown. If an ASM point fell before the origin, the origin would be used as the aim point. The aim points 9, 10, and 11 would be computed using the LAUNCH subroutine.

The processing for block 40 is carried out in four separate steps, utilizing the arrays from common /ASMARRAY/. (1) The ASM targets are first examined to see if they are in range of the origin or some other prior fly point such as a drop bomb point. Those which are not are flagged by setting the corresponding cell of array IFLY to 1. In the example in figure 99, points 3 and 4 would not be flagged, but points 6 and 7 would. (2) The ASM targets flagged in the last step would again be examined in the second pass to see if any were in range of a previous ASM fly point. In the example shown, point 7 would be in range of ASM target 6. The aim point (No. 10) for ASM target 6 is computed at this step. The bomber's path is now fully determined. (3) Sort indices are now generated for all events. For all fly points, the point number is taken as the sort index. For all other points, (i.e., all ASM points which are not fly points; in this example, points 3, 4, and 7), the sort index is a number whose integer part is the earliest point just out of range, and whose fractional part is the distance to this point. After the sort indices are generated, the list is approximately rearranged using the ORDER and REORDER functions. (4) The aim points for the rest of the ASMs are calculated.

Subroutine FLYPOINT, which has the entries PREFL1, PREFL2, and POSTFLY, is logically an integral part of this coding block. Subroutine LAUNCH is called only by POSTFLY.

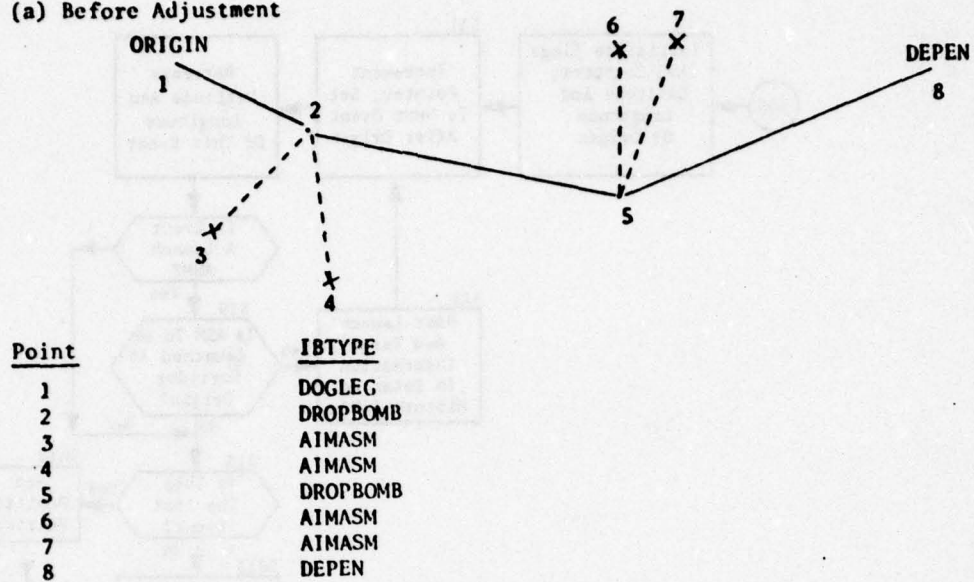
If the last event is an ASM event, the depenetration corridor is re-selected.

The locations of appropriate Change Altitude events associated with the ranges G_2 and G_3 are now calculated by subroutine ADJUST. If the target area was found to be degenerate, blocks 50 and 60 are skipped.

Block 50: Apply GOLOW-2 Before First Target (figure 100)

Block 50 posts all events in the target area, including ASM launches from the corridor origin, and altitude changes. All events except ASM launches are entered in the detailed History table as one-line events. For ASM launches two lines are required, the first for information pertaining to the launch and the second for information pertaining to the target. As the event list is processed, all possible Decoy

(a) Before Adjustment



(b) After Adjustment

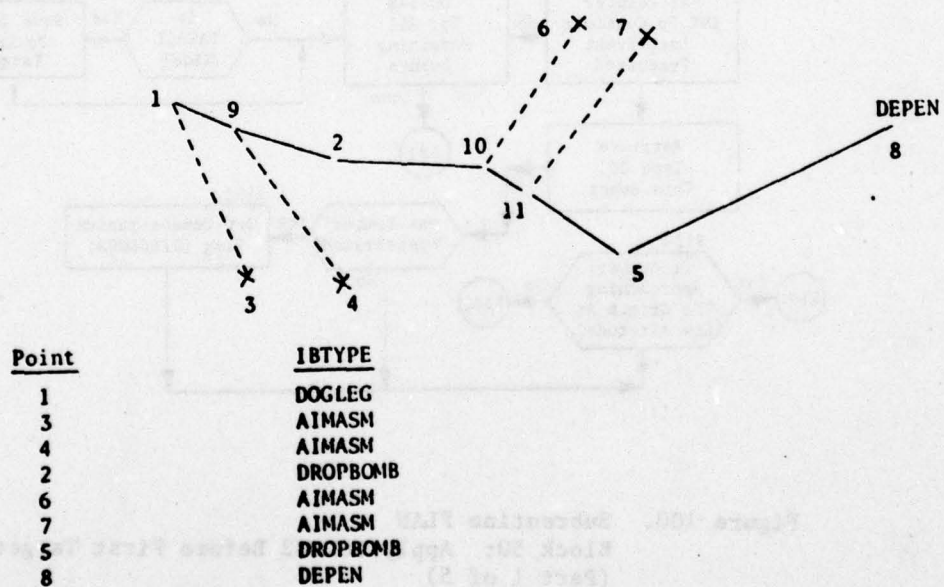


Figure 99. Illustration of ASM Event Adjustment

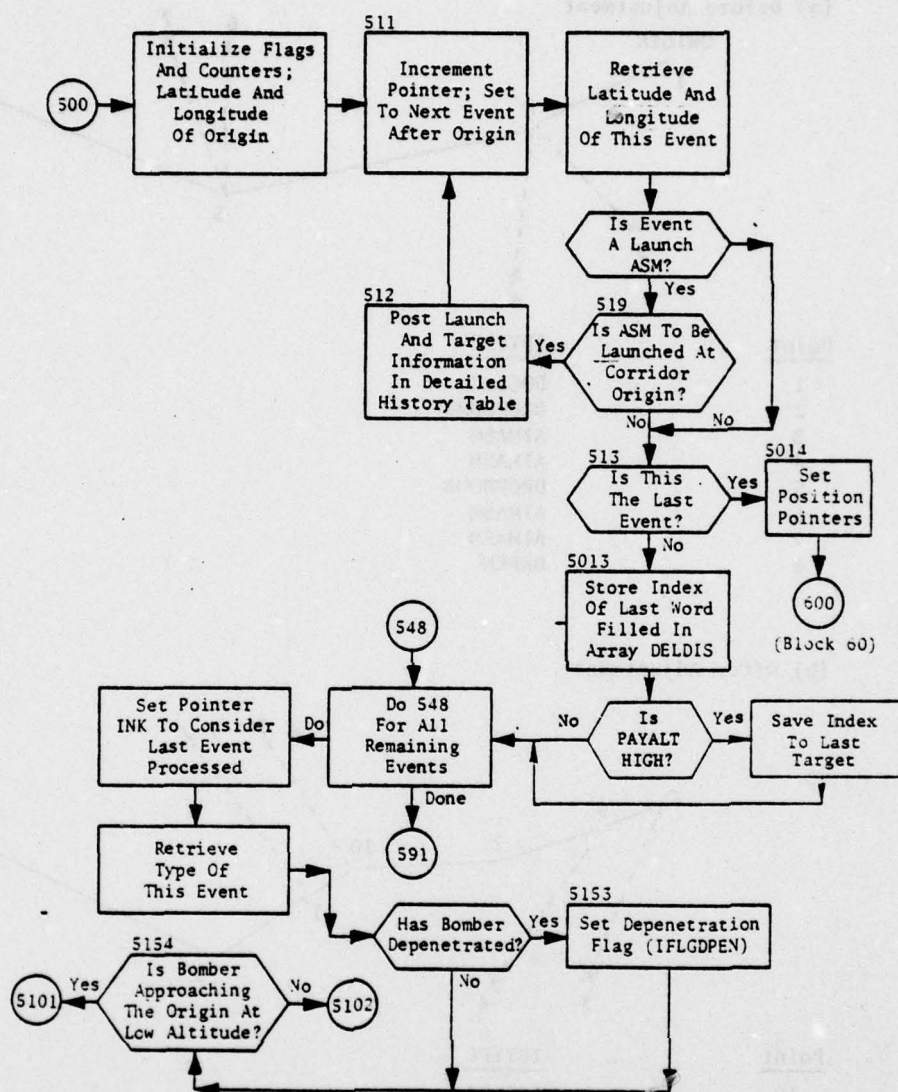


Figure 100. Subroutine PLAN
Block 50: Apply GOLOW2 Before First Target
(Part 1 of 5)

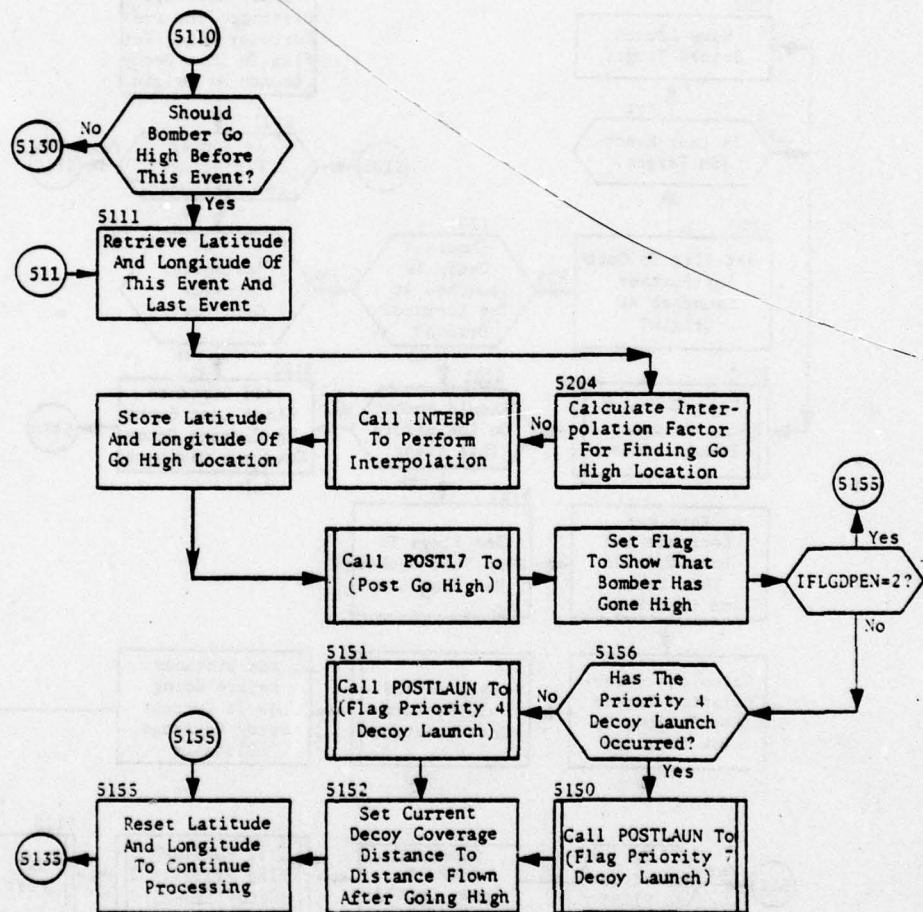


Figure 100. (Part 3 of 5)

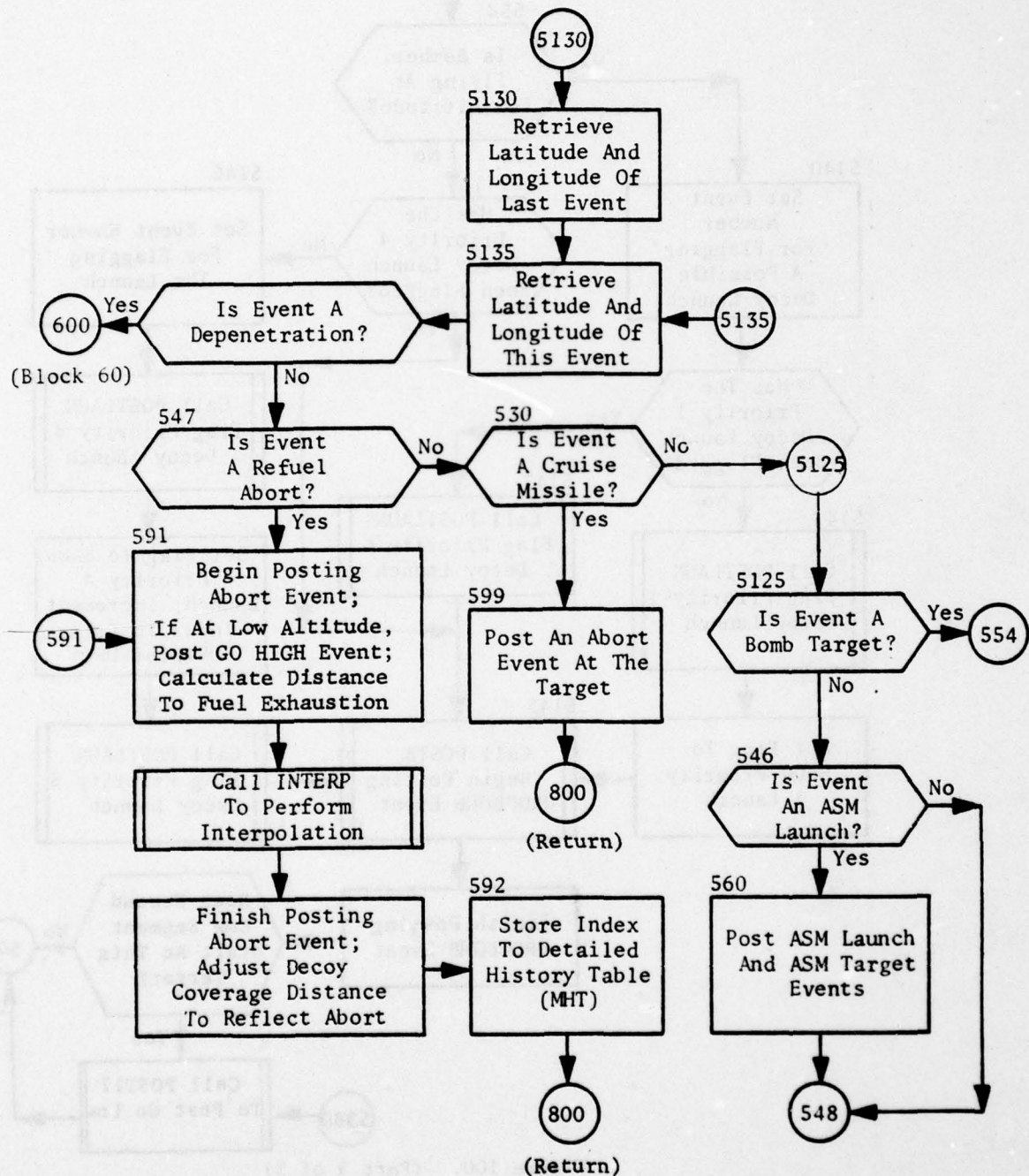


Figure 100. (Part 4 of 5)

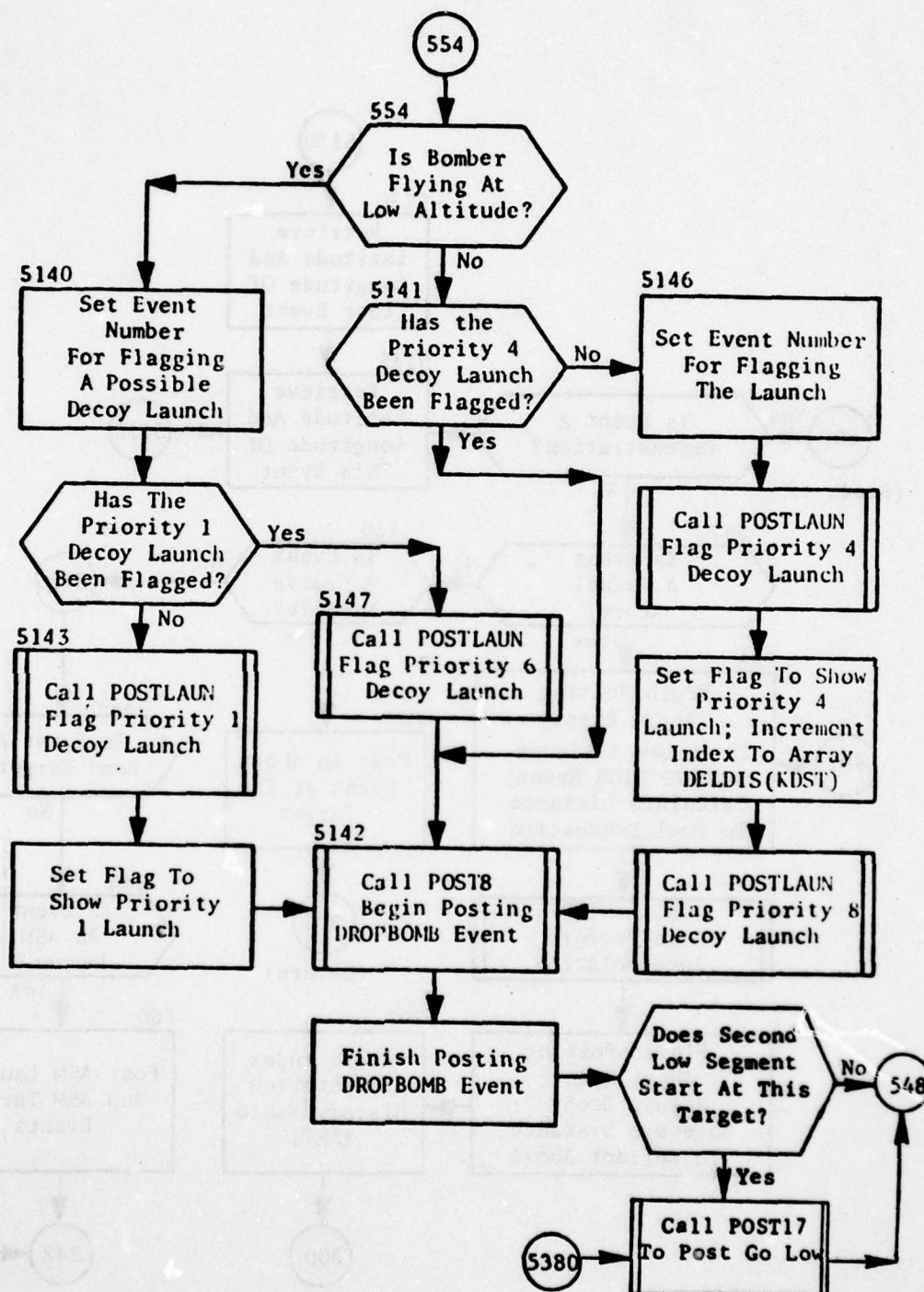


Figure 100. (Part 5 of 5)

Launch situations are flagged by storing the appropriate information pertaining to the launch and the second for information pertaining to the target. As the event list is processed, all possible Decoy Launch situations are flagged by storing the appropriate information in arrays LMHT, LPRIORITY, and NDCYRQ (see block 30). If the value of attribute PAYALT is HIGH, this block checks if a second segment of low-altitude flight is planned after the last target. If so, an altitude change event is placed after the last target.

Processing is terminated with the occurrence of the input events DEPEND, LAND, or DIVEMISL. For DEPEND, a normal exit is made to block 60 described in the next section. For LAND, post a GO HIGH event if currently at low altitude and determine an abort point on the line connecting the last target and the depenetration point at which the bomber's range (fuel) is exhausted. If the bomber would normally use all available range in reaching its last assigned target, a hypothetical abort point is established 5 minutes' flight distance beyond the target. For DIVEMISL, an abort event is posted immediately at the last target.

Block 60: Post Depenetration Events (figure 101)

This block of coding completes the processing for a normal sortie, processing from the last target to the recovery point or base. It computes the most distant recovery base that has available space, associated with this depenetration point, that the bomber can reach. The information on this base and the depenetration corridor index are recorded in the depenetration event. Counts are updated and saved within arrays NAMCAP and NUSED for eventual summarizing prints. In addition, the time of flight to each of the possible recovery bases is computed and stored. These calculations follow the processing of the depenetration leg events.

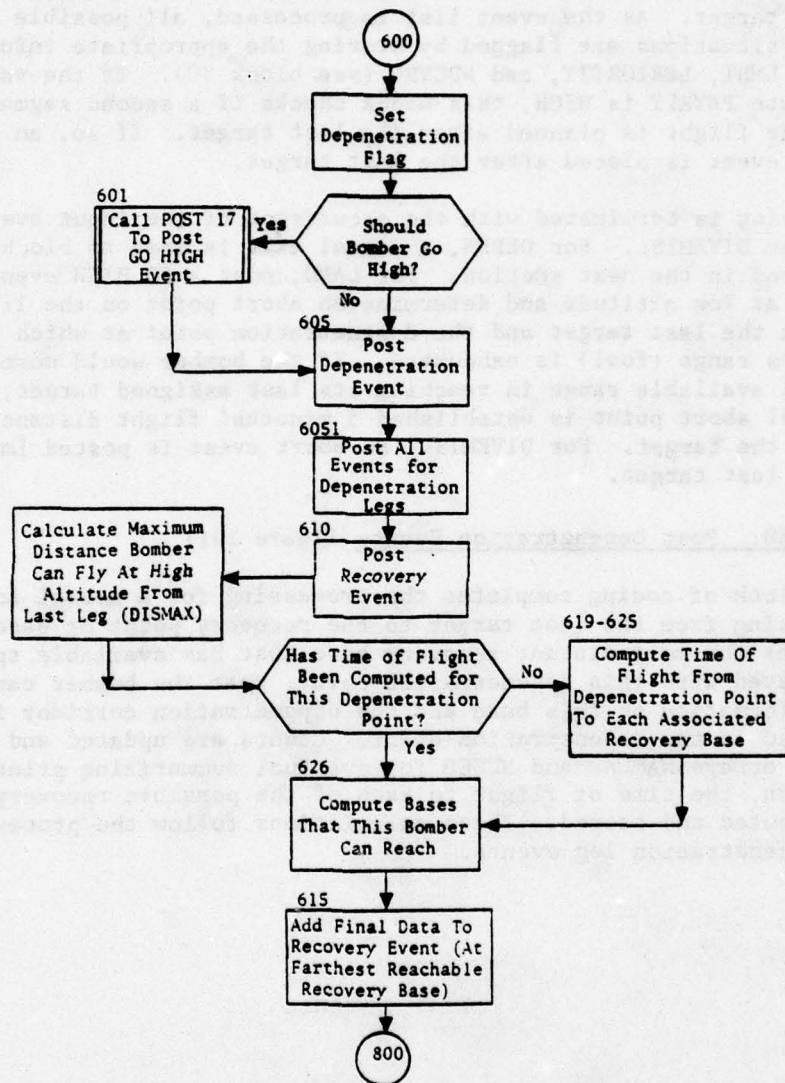


Figure 101. Subroutine PLAN
Block 60: Post Depenetration Events

4.8.14 Subroutine PLANBOMB

PURPOSE: Control processing of bomber plans

ENTRY POINTS: PLANBOMB

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, DECA, DINDATA, DINDT2, EVENTS, MH2, OUTSRT, POLITE, PPINFO, PPXX, RL

SUBROUTINES CALLED: DECOYADD, DISTF, DISTIME, INTERP, PLAN, SNAPIT, SWTCHALT

CALLED BY: ALTPLAN, PLNTPLAN

Method:

The first action of this subroutine is to call subroutine PLAN. Together these two subroutines control the processing of bomber plans. In the discussion that follows, the two subroutines are to be thought of as sequential parts of an integral process.

Figure 102 shows a typical path a bomber would take between the time of its launch and its recovery. The bomber is launched from a base, flies to a refuel point or area if refueling is called for, then to a corridor entry point. It may then fly one or more prespecified doglegs (called precorridor legs) which define a penetration route before reaching the point labeled Corridor Origin. From the origin it flies over the target area and its assigned targets in their proper order. It then enters the depenetration corridor which may also consist of one or more doglegs. From there it flies to the recovery point or base.

This path may logically be divided into four parts: (1) the launch and refuel portion, (2) the precorridor legs, (3) the target area which is the main part of the plan, and (4) the depenetration and recovery portion.

In PLANBOMB/PLAN, each bomber sortie is processed in much the same order as it is flown; that is, first the precorridor section events are posted, then those of the target section, and finally, the depenetration and recovery section events. Besides the posting of the target events themselves, the main processing consists of posting events for changes of altitude and decoy launches. All postings for bomber events are made in the arrays of common /DINDATA/. The completed plan is output, and processing begins on the new plan.

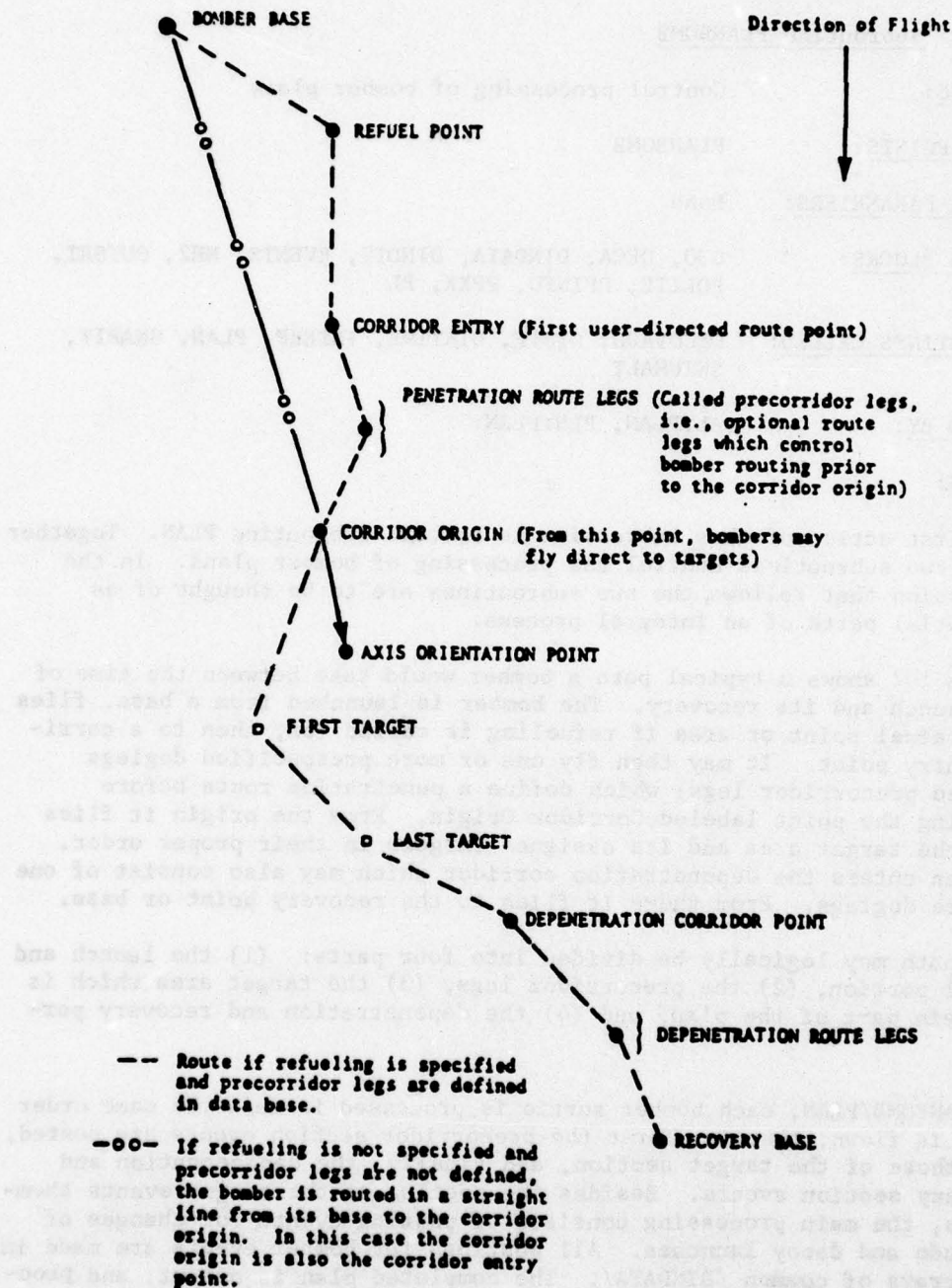


Figure 102. Path of Typical Bomber Sortie

After subroutine PLAN has completed subroutine SWTCHALT is called to convert the CHANGALT events to GO LOW or GO HIGH events. Subroutine DISTIME then is called to compute distances between events and associated time increments, and subroutine DECOYADD is called to allocate the available decoys. Decoy Launches are now added to the detailed History table by examining each event to see if a launch is to be inserted (indicated if the corresponding word in array ILAUNDEC is nonzero). For low-altitude launches (ILAUNDEC=0), the actual launch point must be computed. The Decoy Launches are inserted by copying each event into a temporary detailed History table. If a GO HIGH event has a decoy launch indicated, the launch is inserted after the GO HIGH. For all other events with indicated decoy launches, the launch is inserted before the event is copied. Decoy launches are posted by adding to event LAUNDCOY to the event array (JTP) and storing the number of decoys launched (>0) in the array usually reserved for the place index (KPL). The remaining information required in the detailed History table is stored in the normal manner.

Decoys are terminated as the detailed History table is recopied into its original arrays. Each time a high-altitude Decoy Launch event is encountered, the total decoy flight time is computed from the distance in array DISTORE (filled by subroutine DECOYADD) and added to the next odd word in an array (TSTORE) which holds the remaining flight time of all decoys which have been launched but not yet terminated at the time of this event. The number of decoys to be terminated is added to the next even word of TSTORE. As each subsequent event is processed, the time since the last event (HDT) is subtracted from the times in TSTORE. Whenever a decoy has no flight time remaining, a LAUNDCOY event, together with the number of decoys being terminated (stored as a negative number) and other relevant information, is added to the detailed History table. If the bomber depenetrates or aborts while decoys are still flying, the remaining decoys are terminated immediately before the final event. It should be noted that decoys launched at low altitude are not terminated.

Subroutine PLANBOMB is illustrated in figure 103.



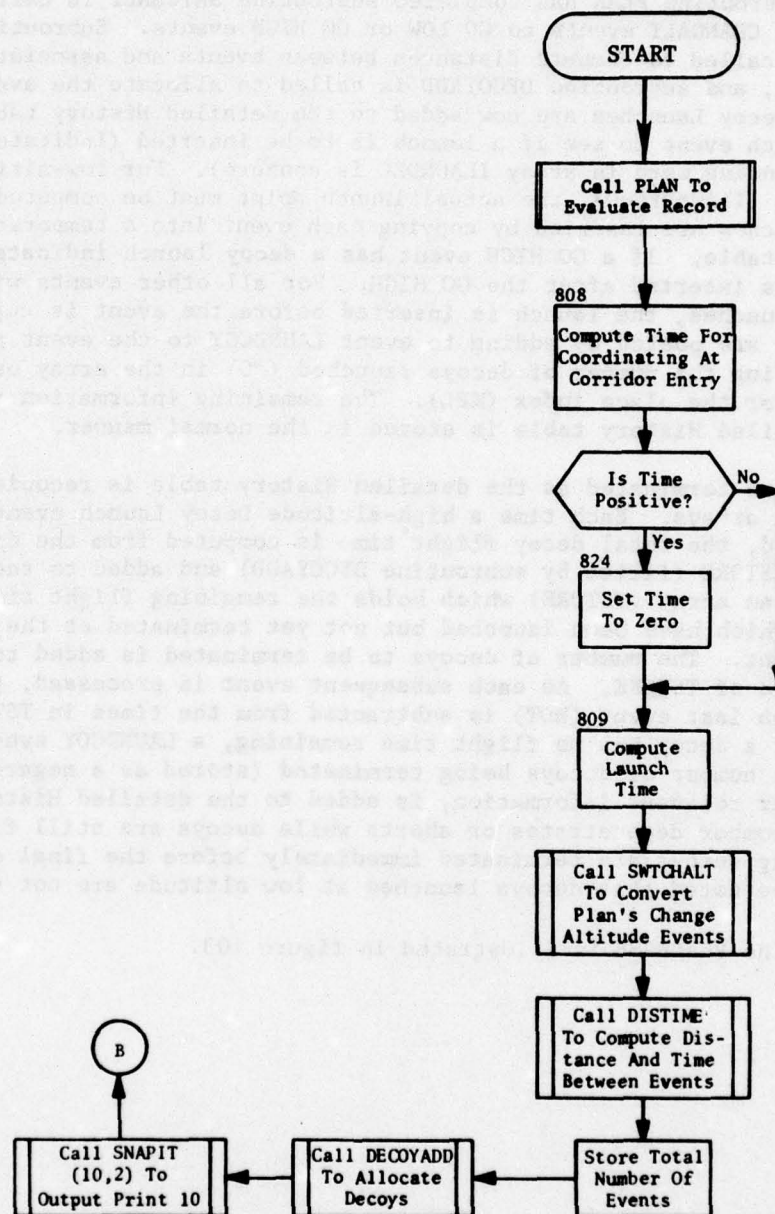


Figure 103. Subroutine PLANBOMB (Part 1 of 9)

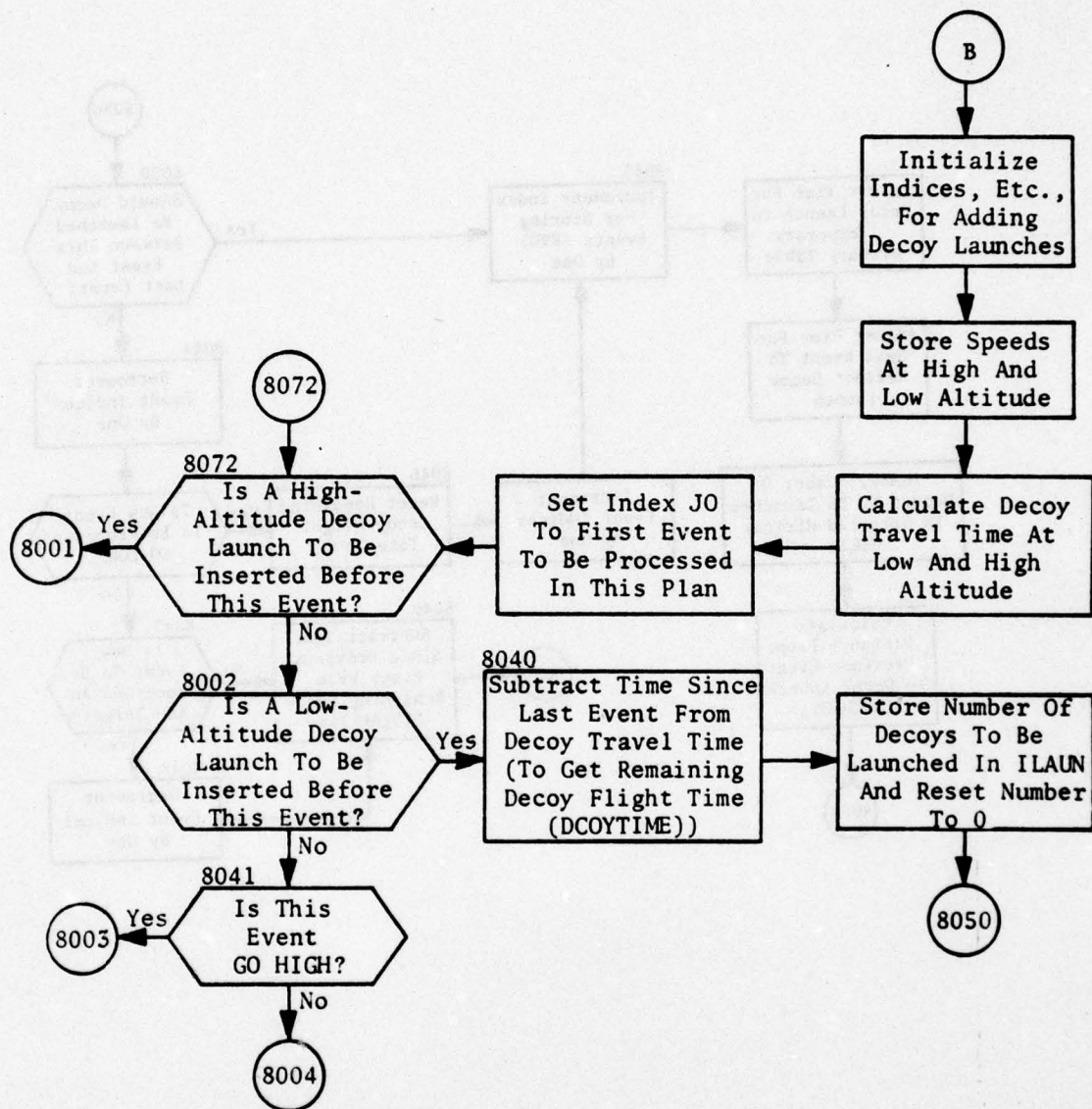


Figure 103. (Part 2 of 9)

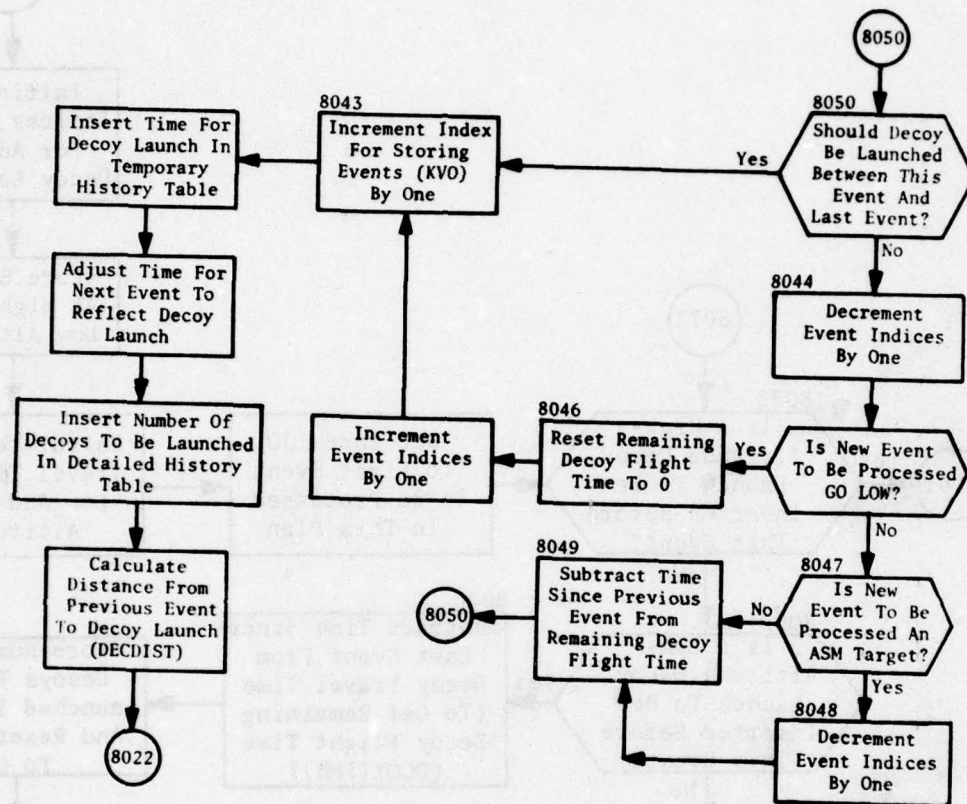


Figure 103. (Part 3 of 9)

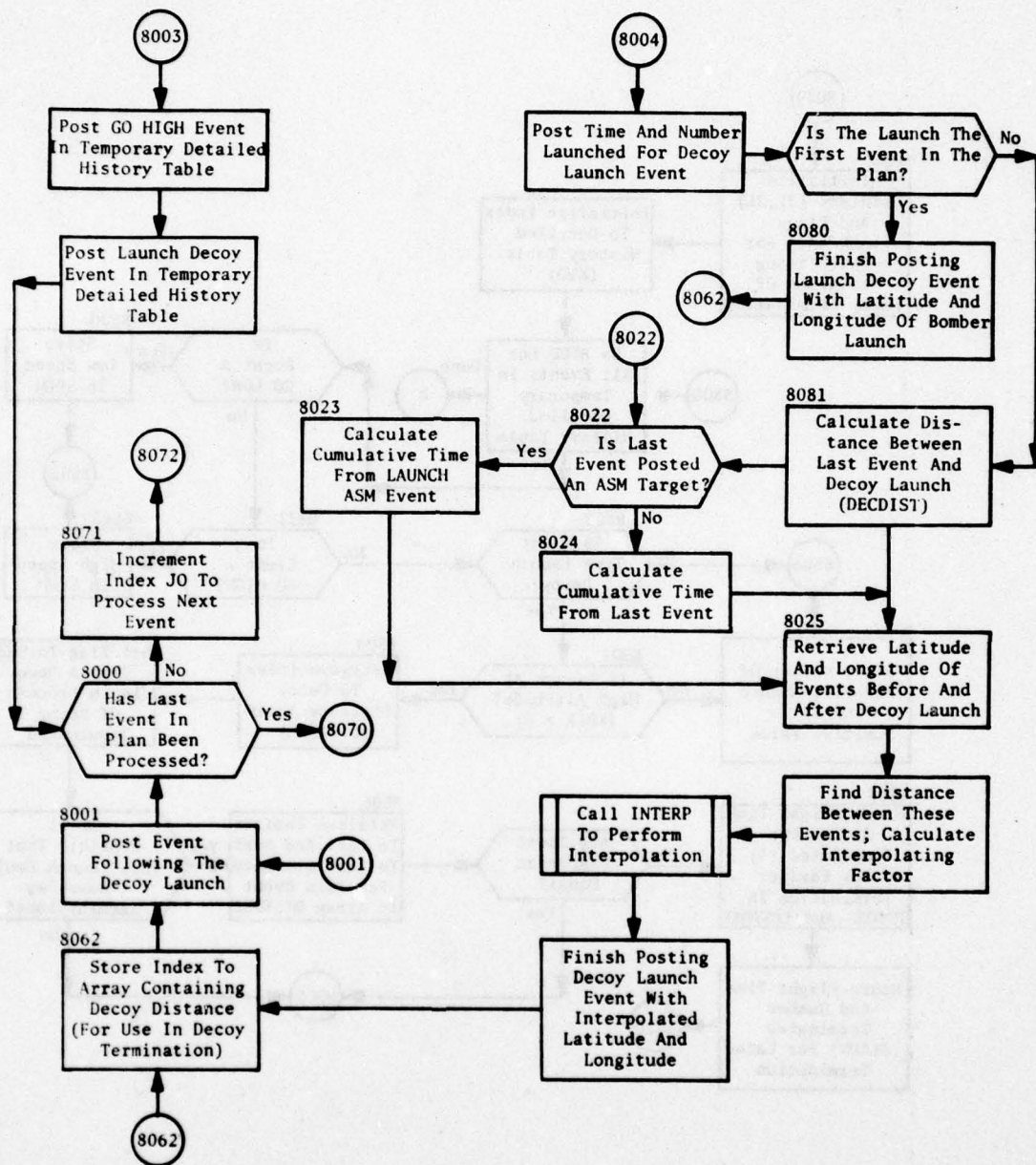


Figure 103. (Part 4 of 9)

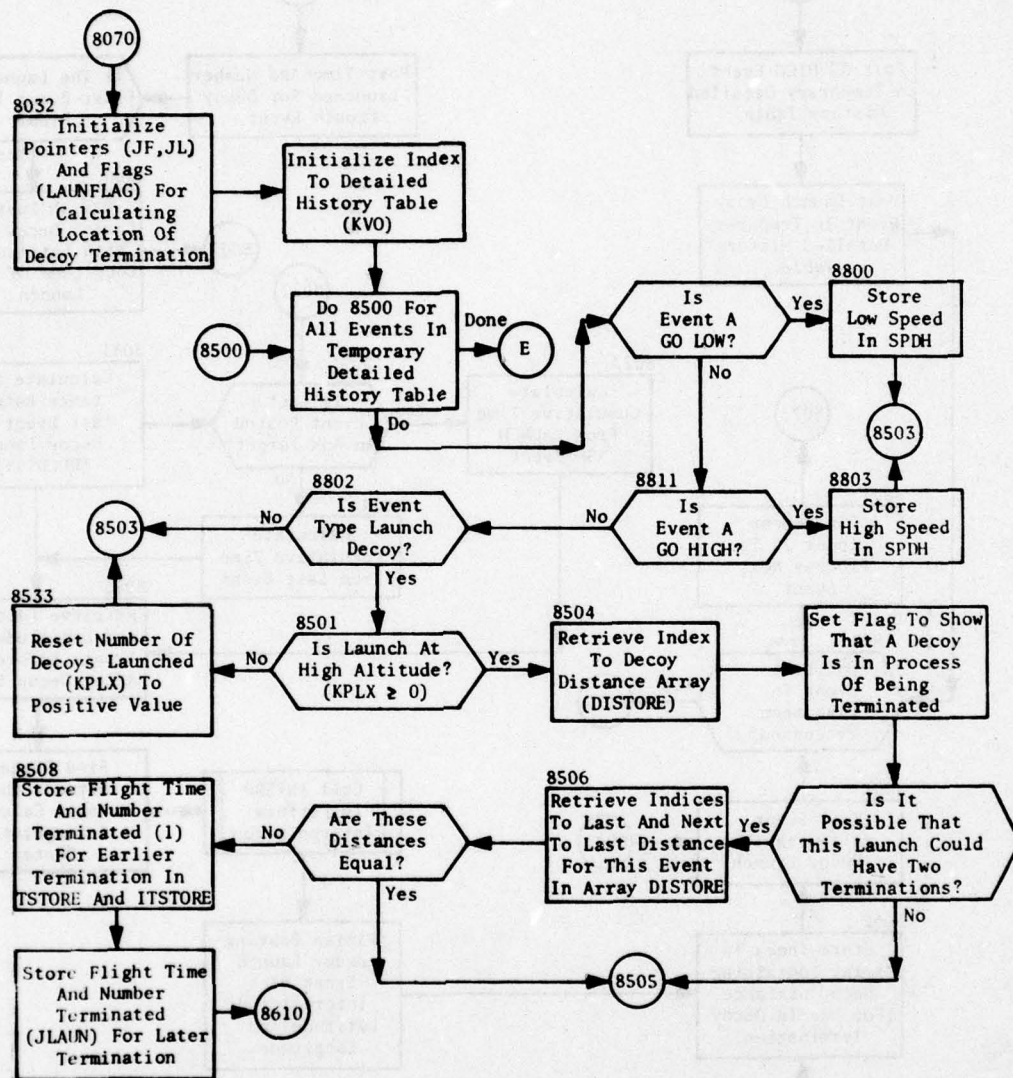


Figure 103. (Part 5 of 9)

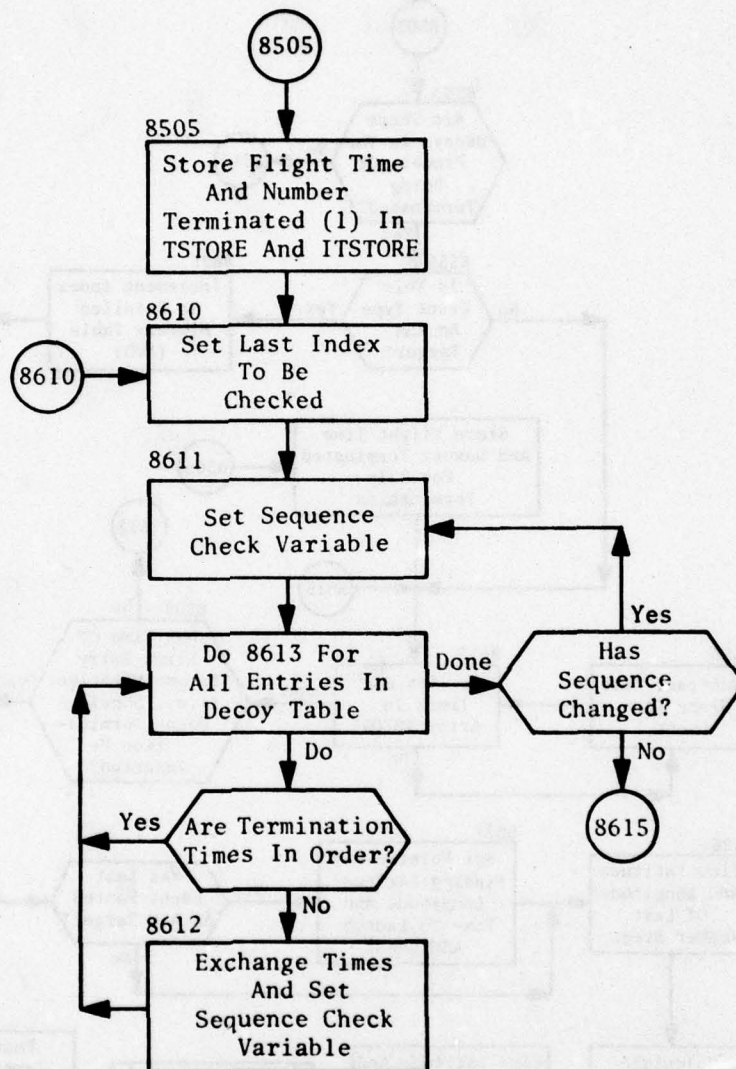
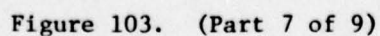


Figure 103. (Part 6 of 9)



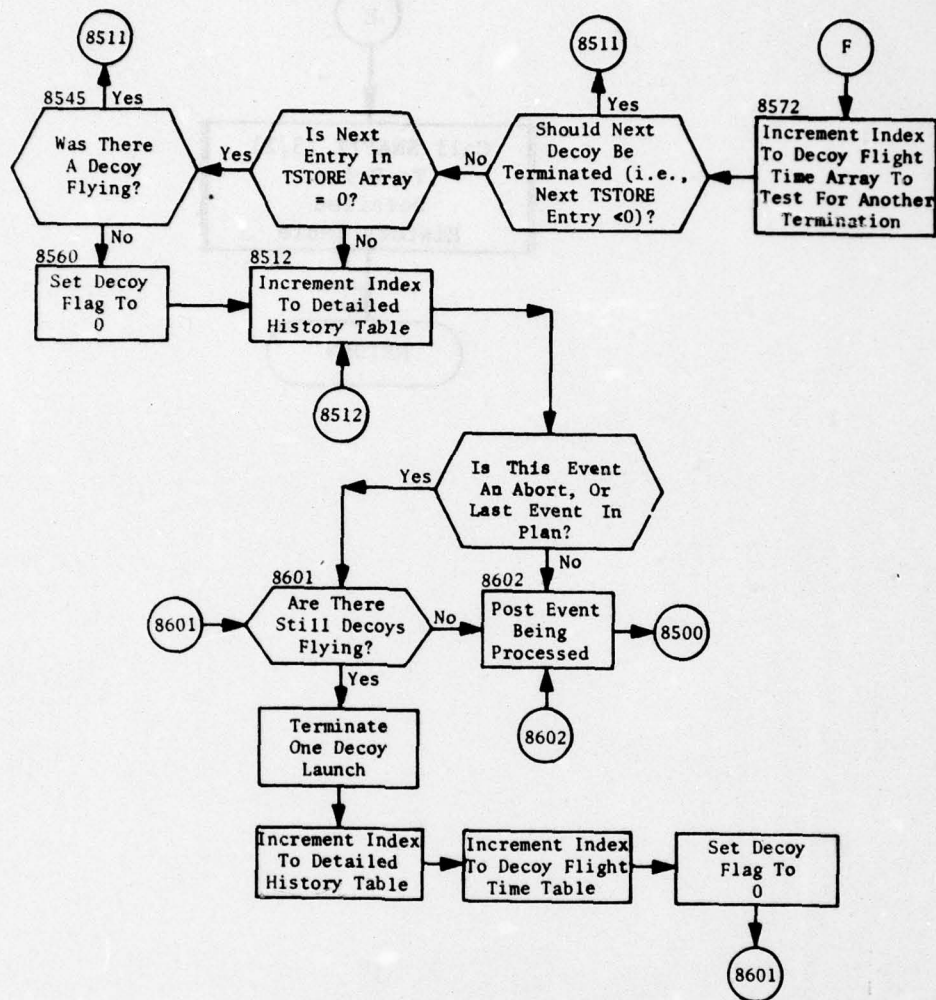


Figure 103. (Part 8 of 9)

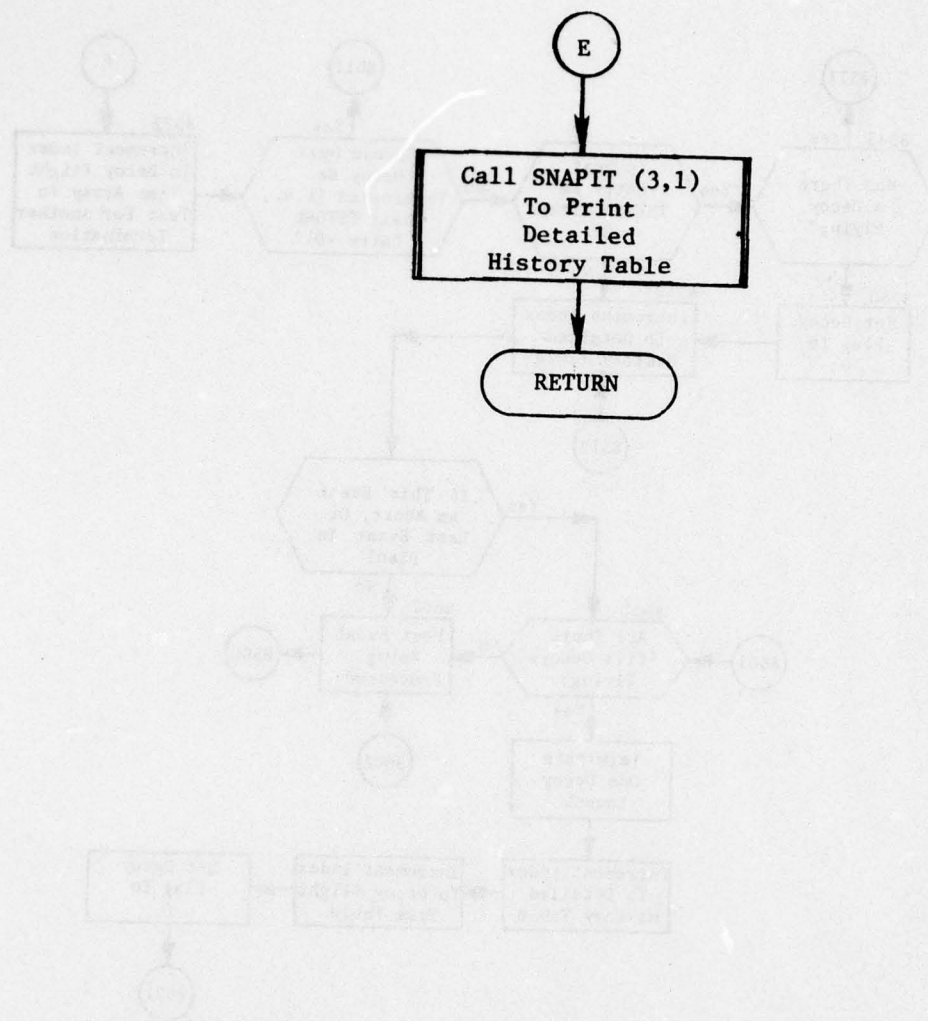


Figure 103. (Part 9 of 9)

4.8.15 Subroutine PLANTMIS

PURPOSE: Control processing of missile plans.

ENTRY POINTS: PLANTMIS

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, GRPSTF, TIMELINE, TYPSTF

SUBROUTINES CALLED: ATN2PI, DISTF, GLOG, HEAD, MODIFY, NEXTTT

CALLED BY: ALTPLAN, PLNTPLAN

Method:

The subroutine's main function is to determine the missile launch time based on any timing information provided by the user via MISTME and MSLCOR clauses (subroutine LNCHDATA). First the subroutine checks through all the target assignments. This information is needed since, if there are several targets assigned to a missile and more than one have fixed time assignments, only the first fixed time assignment encountered will be considered. Thus, if a previous fixed time assignment has determined the launch time for the missile, no further calculations need be done to compute the launch time for later reentry vehicles on the missile. If there are no fixed assignments (with timing) on a missile with MIRV payload, the launch time is computed by considering only the data for the target assigned to the first reentry vehicle on the booster.

For salvoed missiles, local parameter DELTA is calculated based on salvo number, number of simultaneous launches, and launch interval. DELTA is the amount added to the basic launch time for launch interval constraints.

If the weapon is not fixed, PLANTMIS checks the plan type. If the strike is retaliatory (INITSTRK = 2) the complicated time plan is ignored and the launch time is the time specified by FOOTPRNT. If INITSTRK = 1 there are two options. If the missile type has a FLIGHT CORMSL the launch time is computed so that the fraction of the flight specified by CORMSL is completed at time zero. If the missile type has a LINE CORMSL the situation is more complex.

The subroutine then calculates whether the missile flight path crosses one of the timing lines input to subroutine LNCHDATA. If the missile crosses a line, the launch time is computed so that the missile crosses the timing line at time equal to CORMSL. If the missile fails to cross any line, the launch time is chosen so that the missile will impact at time zero.

Finally, the launch time is stored in attribute SLOW1 and the sortie table (SRTYTB) modified.

Subroutine PLANTMIS is illustrated in figure 104.

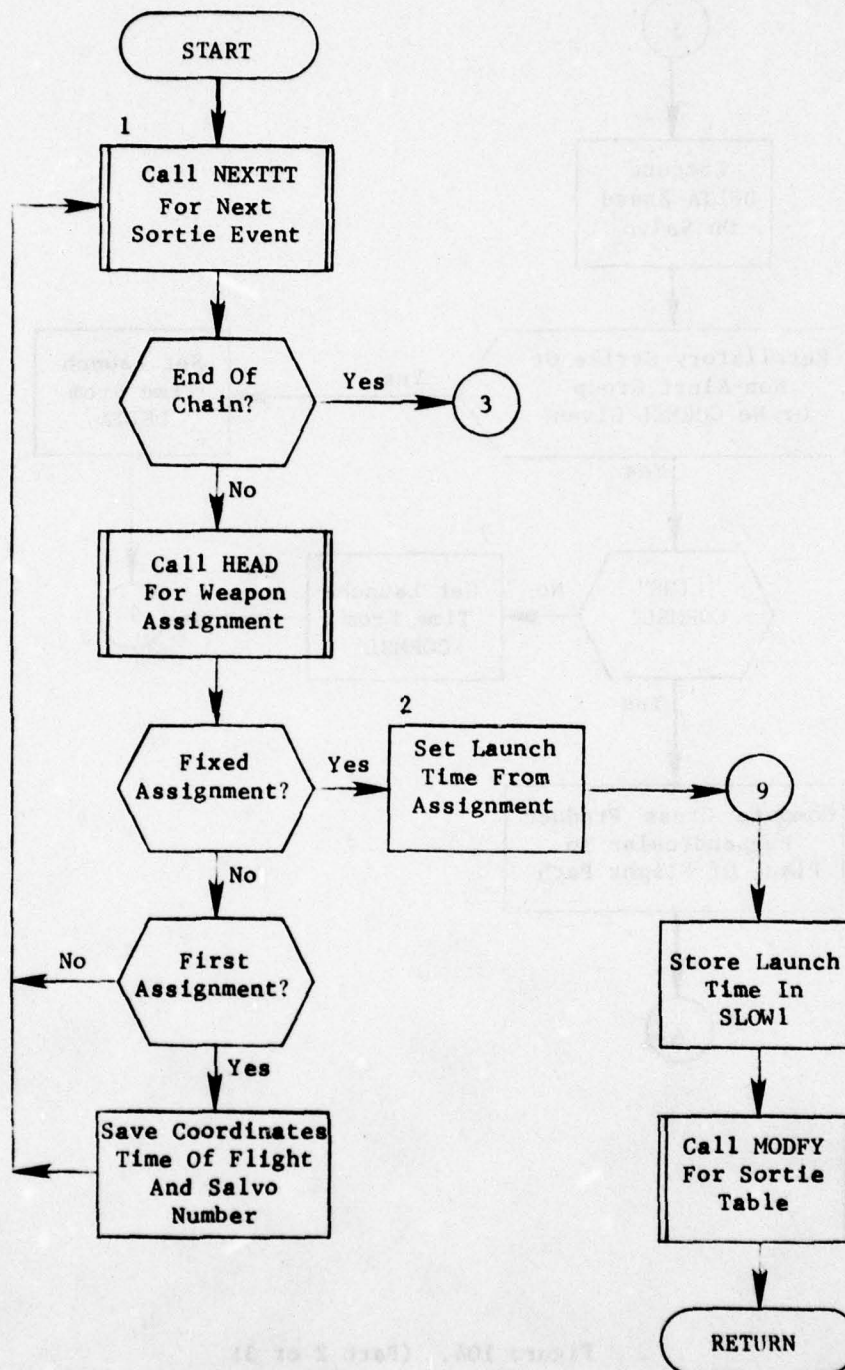


Figure 104. Subroutine PLANTMIS (Part 1 of 3)

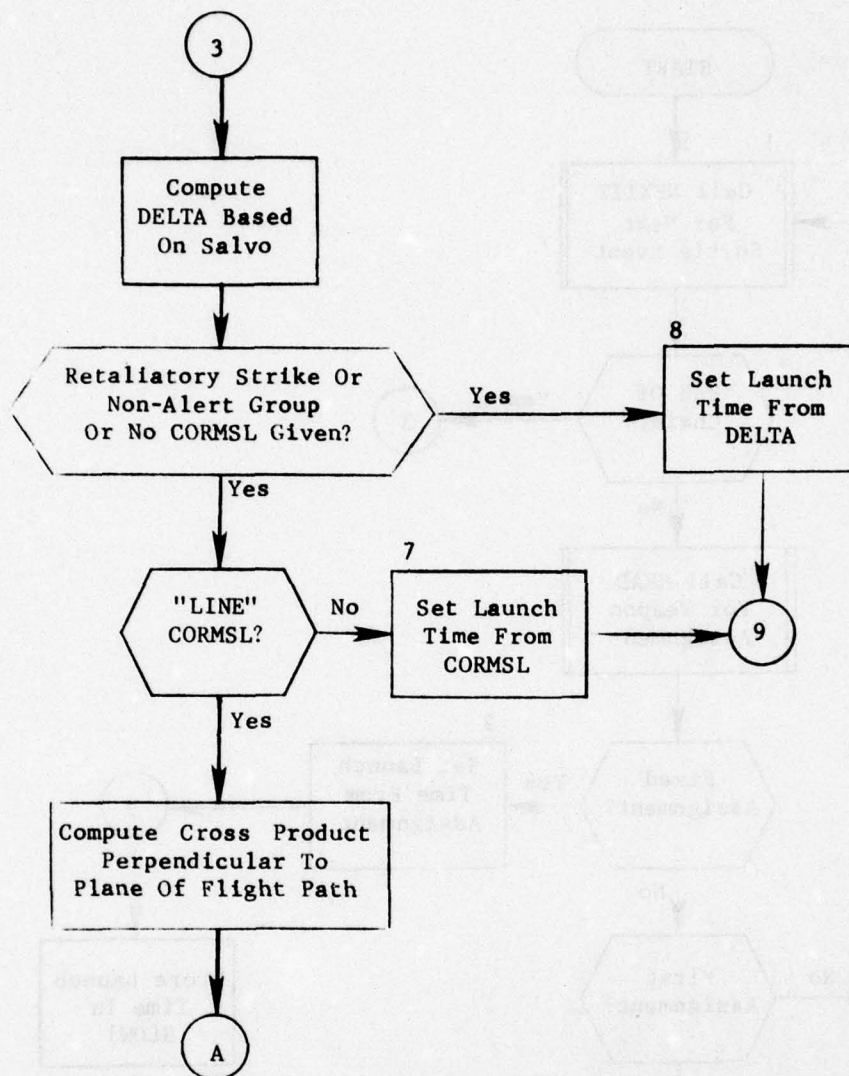


Figure 104. (Part 2 of 3)

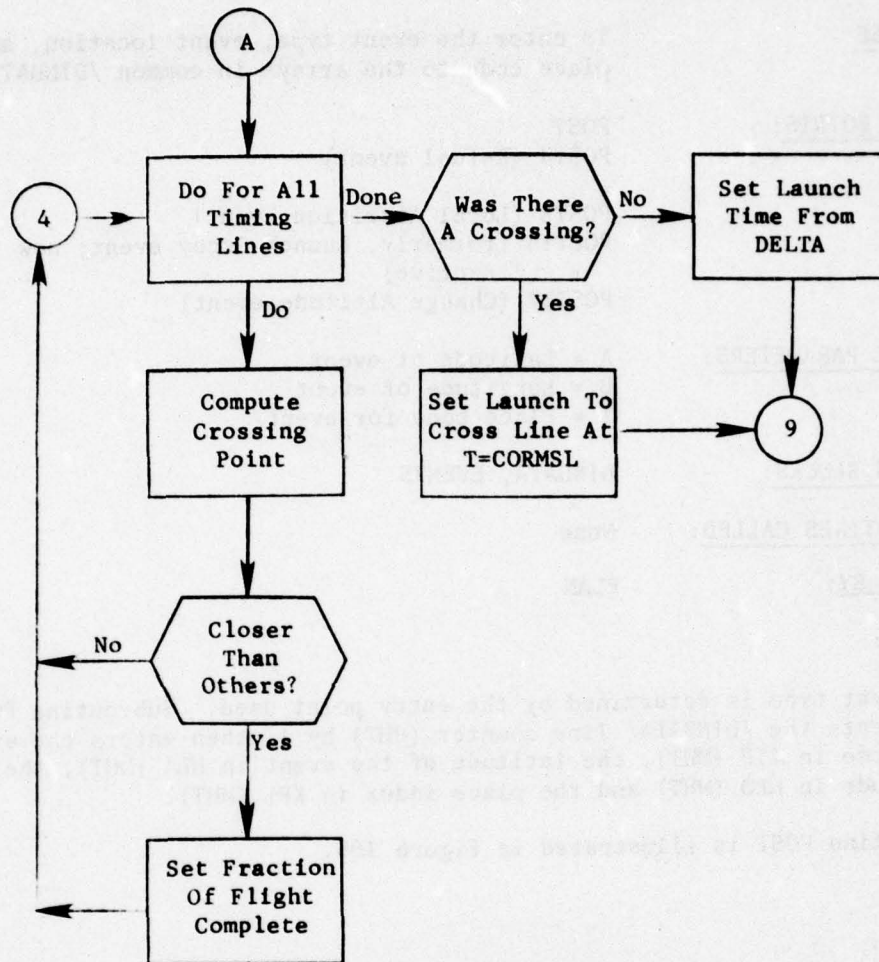


Figure 104. (Part 3 of 3)

4.8.16 Subroutine POST

PURPOSE: To enter the event type, event location, and place code to the arrays in common /DINDATA/.

ENTRY POINTS: POST
POST4 (Refuel event)
POST8 (Local Attrition event)
POST15 (formerly, Launch Decoy event; now inactive)
POST17 (Change Altitude event)

FORMAL PARAMETERS: A = Latitude of event
B = Longitude of event
I = Place code for event

COMMON BLOCKS: DINDATA, EVENTS

SUBROUTINES CALLED: None

CALLED BY: PLAN

Method:

The event type is determined by the entry point used. Subroutine POST increments the /DINDATA/ line counter (MHT) by 1, then enters the event type code in JTP (MHT), the latitude of the event in HLA (MHT), the longitude in HLO (MHT) and the place index in KPL (MHT).

Subroutine POST is illustrated in figure 105.

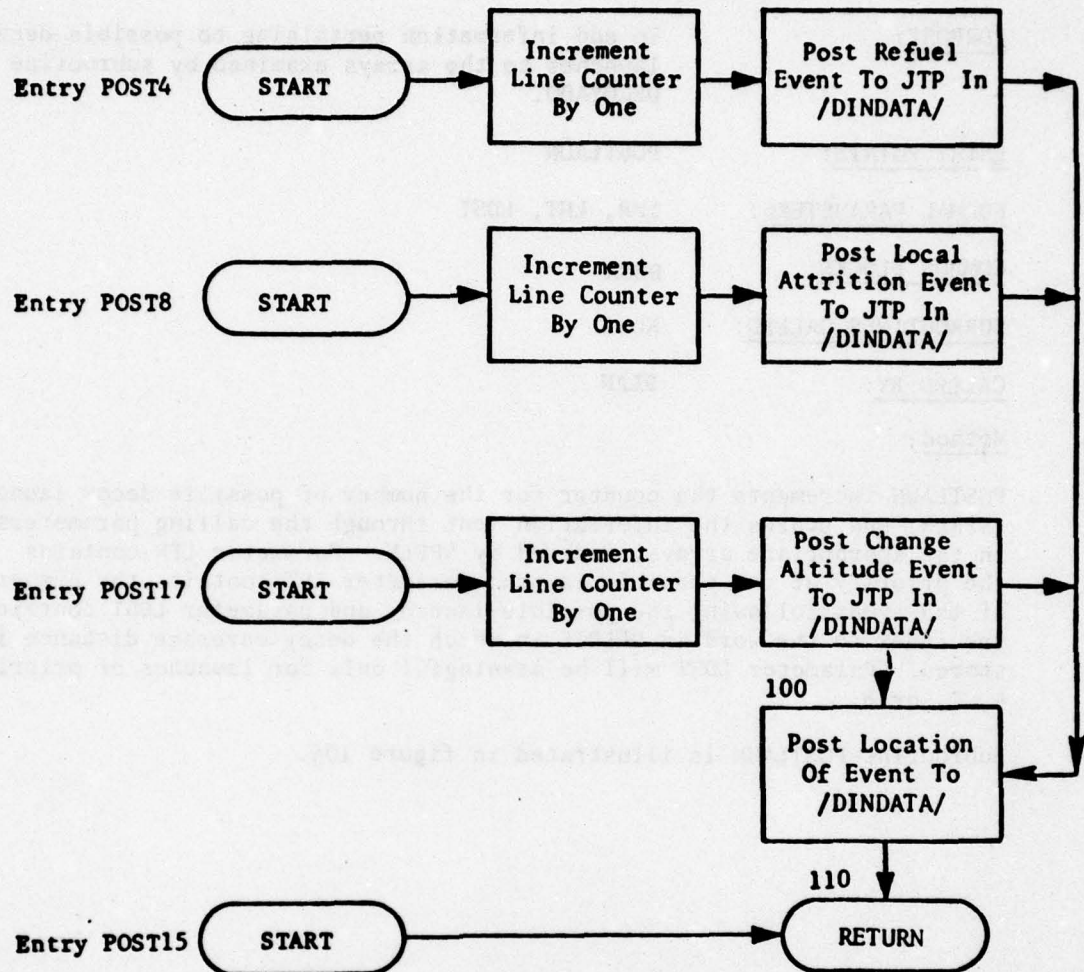


Figure 105. Subroutine POST

4.8.17 Subroutine POSTLAUN:

<u>PURPOSE:</u>	To add information pertaining to possible decoy launches to the arrays examined by subroutine DECOYADD.
<u>ENTRY POINTS:</u>	POSTLAUN
<u>FORMAL PARAMETERS:</u>	LPR, LHT, LDST
<u>COMMON BLOCKS:</u>	DECA
<u>SUBROUTINES CALLED:</u>	None
<u>CALLED BY:</u>	PLAN

Method:

POSTLAUN increments the counter for the number of possible decoy launches (NPSLN) and stores the information sent through the calling parameters in the appropriate arrays, indexed by NPSLN. Parameter LPR contains the priority of the possible launch; parameter LHT contains the number of the event following the possible launch; and parameter LDST contains the index to the word in DELDIS in which the decoy coverage distance is stored. Parameter LDST will be meaningful only for launches of priority 5, 7, or 8.

Subroutine POSTLAUN is illustrated in figure 106.

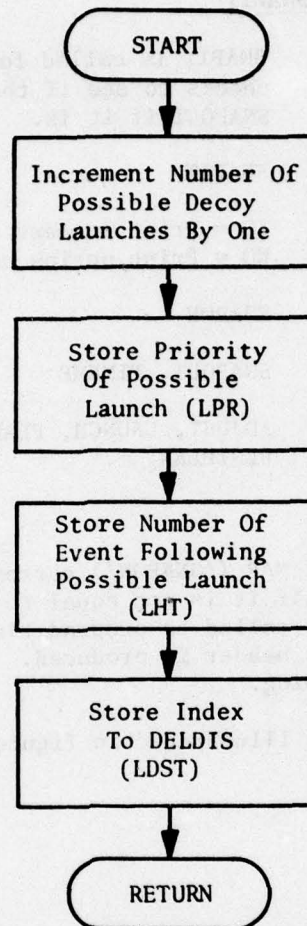


Figure 106. Subroutine POSTLAUN

4.8.18 Subroutine SNAPIT

PURPOSE: SNAPIT is called for all optional prints. It checks to see if the print is active and calls SNAPOUT if it is.

ENTRY POINTS: SNAPIT

FORMAL PARAMETERS: IO = Print request number
NO = Print option code

COMMON BLOCKS: SNAPON

SUBROUTINES CALLED: SNAPOUT, TIMEME

CALLED BY: ADJUST, LAUNCH, PLAN, PLANBOMB, PLANTANK, PLANTMIS, PLNTPLAN

Method:

First the element of NAP (/SNAPON/) corresponding to the print request number is checked. If it is not equal to 3 the subroutine exits. Otherwise, TIMEME is called to suspend timing. If the print option code is equal to 1 a header is produced. SNAPOUT is called and then TIMEME to resume timing.

Subroutine SNAPIT is illustrated in figure 107.

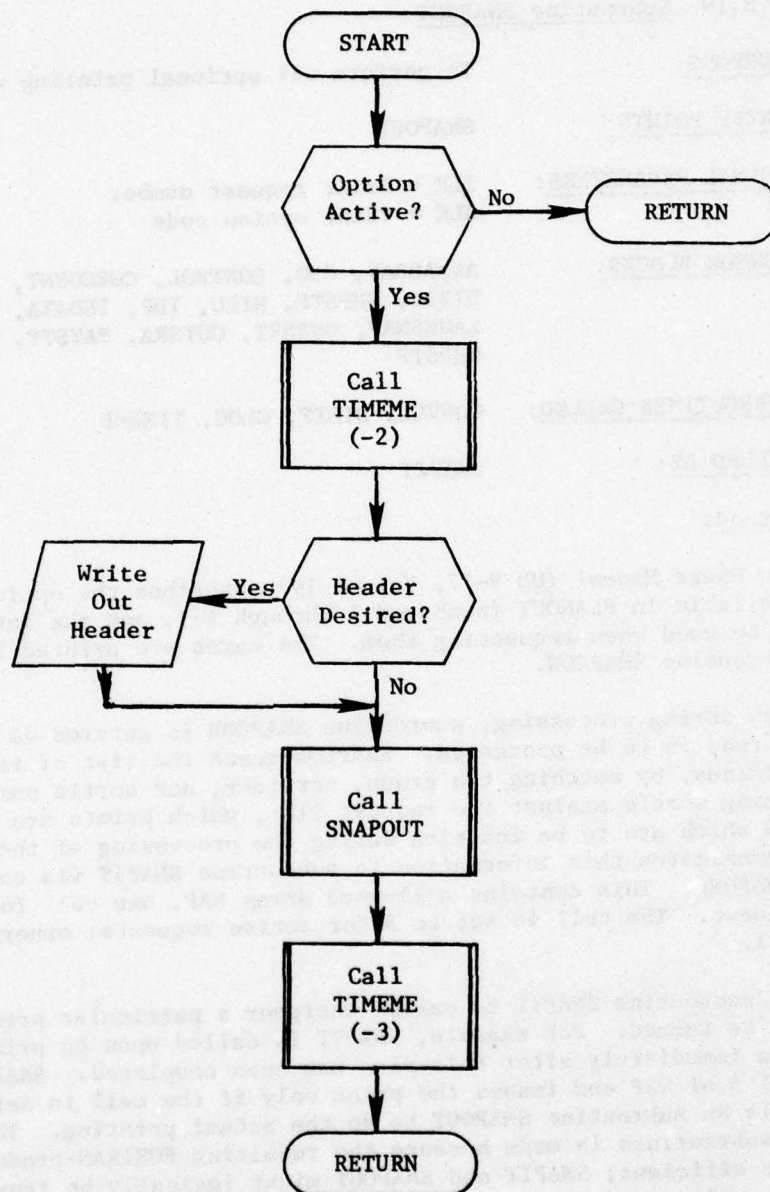


Figure 107. Subroutine SNAPIT

4.8.19 Subroutine SNAPOUT

PURPOSE: To perform all optional printing within PLANOUT.

ENTRY POINTS: SNAPOUT

FORMAL PARAMETERS: ILK = Print request number
MLK = Print option code

COMMON BLOCKS: ASMARRAY, C30, CONTROL, CORCOUNT, DINDATA, DINDT2,
DISTC, GRPSTF, HILO, IDP, INDATA, IOUT, IRF, LASM,
LAUNSNAP, OUTSRT, OUTSRA, PAYSTF, SPASM, TYPSTF,
WHDSTF

SUBROUTINES CALLED: CONVLL, DISTF, GLOG, TIMEME

CALLED BY: SNAPIT

Method:

The Users Manual (UM 9-77, Volume IV) describes the optional prints available in PLANOUT (numbered 1 through 15), and the data card format to be used when requesting them. The cards are printed initially by subroutine SNAPCON.

Then during processing, subroutine SNAPCON is entered as each new sortie is read in to be processed. SNAPCON scans the list of requests and determines, by matching the group, corridor, and sortie numbers of the incoming sortie against the request list, which prints are to be activated and which are to be inactive during the processing of the record. It communicates this information to subroutine SNAPIT via common block /SNAPON/. This contains a 15-word array NAP, one cell for each print request. The cell is set to 3 for active requests; otherwise it is set to 1.

The subroutine SNAPIT is called wherever a particular print might possibly be issued. For example, SNAPIT is called upon to print the detailed plan immediately after this plan has been completed. SNAPIT then checks cell 3 of NAP and issues the print only if the cell is set to 3. It calls on subroutine SNAPOUT to do the actual printing. This separation of subroutines is made because the resulting FORTRAN-produced program is more efficient; SNAPIT and SNAPOUT might logically be treated as one subroutine.

SNAPOUT itself contains only printing routines. In some instances, the print option code (MLK), passed with the print request number, may select differing print options within the given print number.

Subroutine SNAPOUT is illustrated in figure 108.

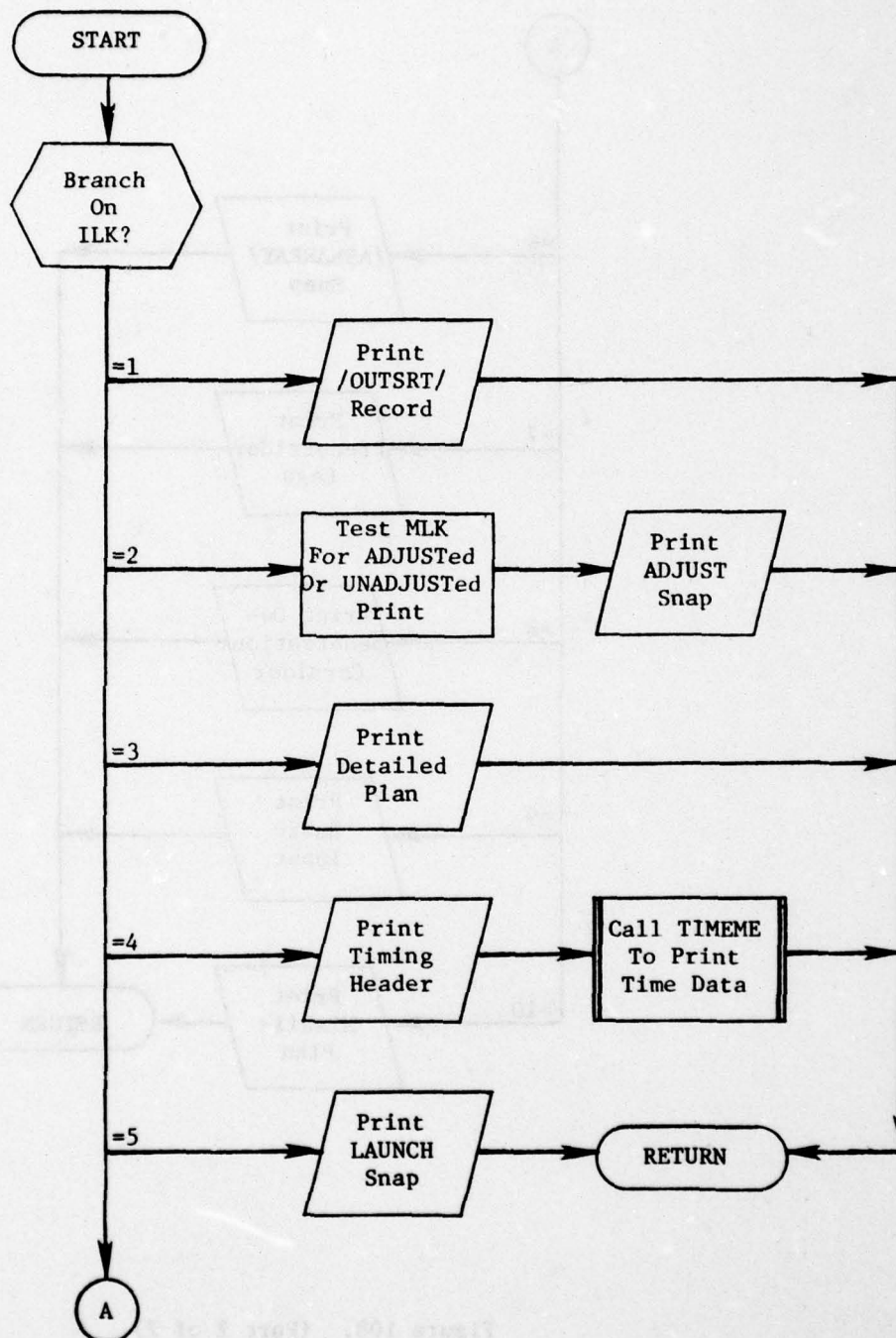


Figure 108. Subroutine SNAPOUT (Part 1 of 2)

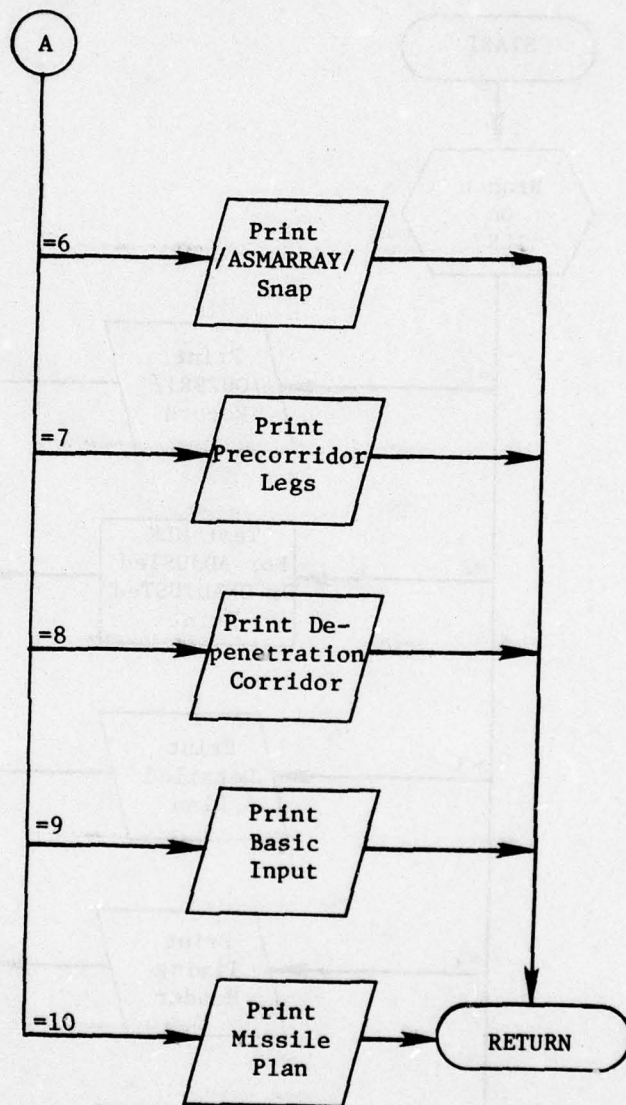


Figure 108. (Part 2 of 2)

SORBOM

None

data is entered
the events cu
s takes place,
the reference

kes the events
s simply saved
t, the appropri
rieved before

illustrated in

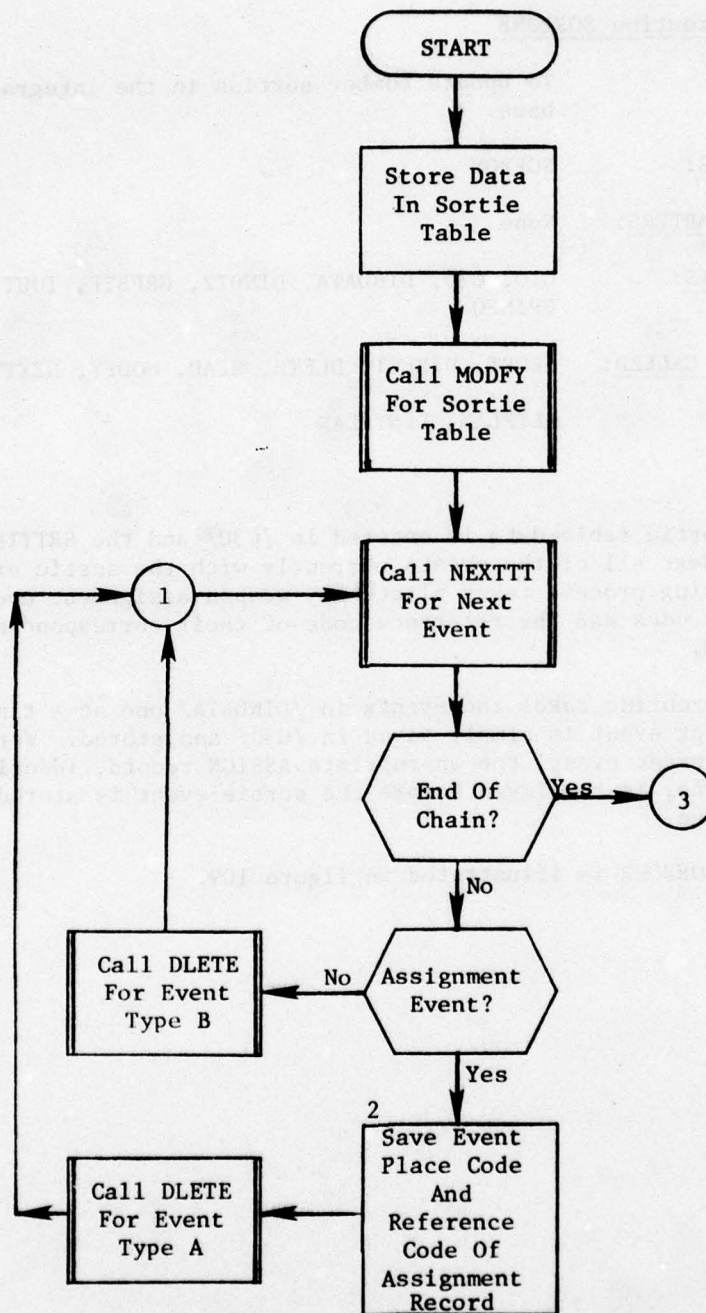


Figure 109. Subroutine SORBOMB (Part 1 of 2)

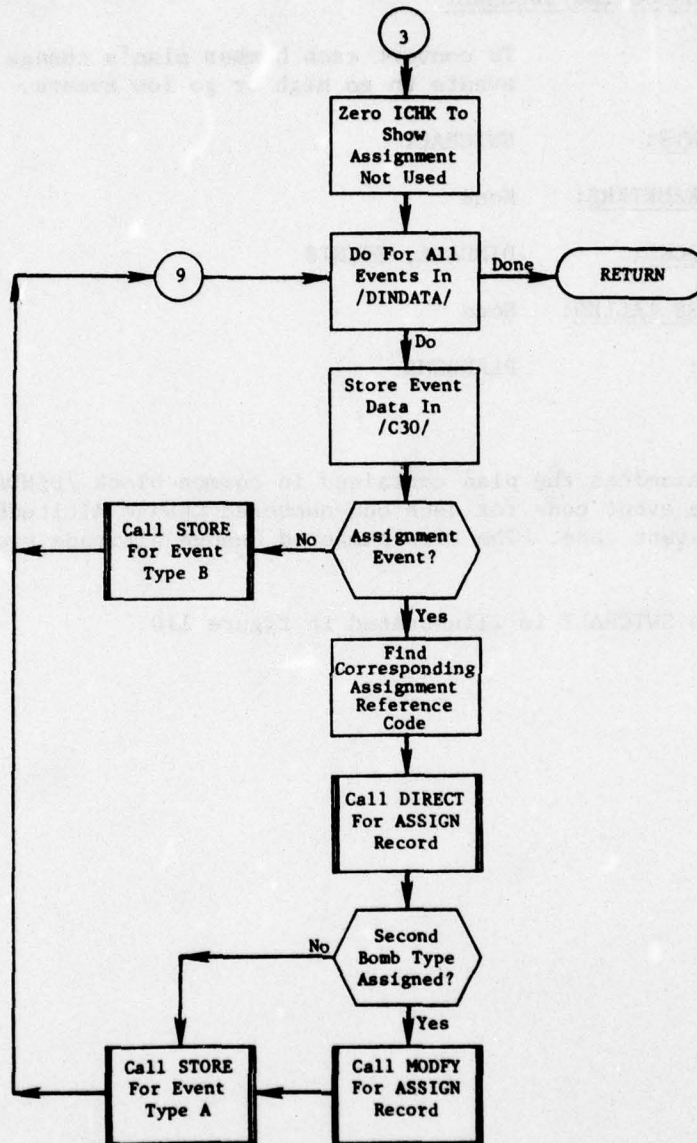


Figure 109. (Part 2 of 2)

4.8.21 Subroutine SWTCHALT

PURPOSE: To convert each bomber plan's change altitude events to go high or go low events.

ENTRY POINTS: SWTCHALT

FORMAL PARAMETERS: None

COMMON BLOCKS: DINDATA, EVENTS

SUBROUTINES CALLED: None

CALLED BY: PLANBOMB

Method:

SWTCHALT examines the plan contained in common block /DINDATA/, and replaces the event code for each odd-numbered Change Altitude event with a go low event code. The even-numbered Change Altitude events become go highs.

Subroutine SWTCHALT is illustrated in figure 110.



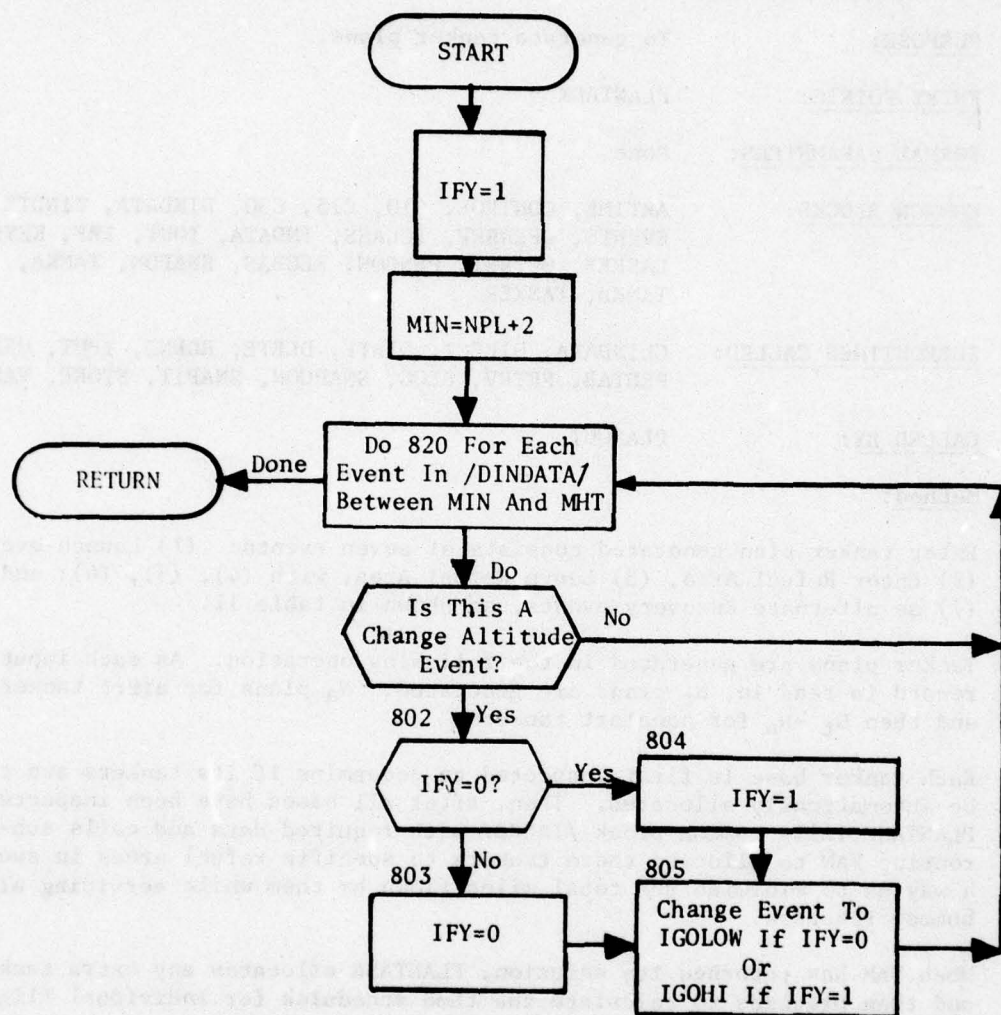


Figure 110. Subroutine SWTCHALT

4.9 Subroutine PLANTANK*

PURPOSE: To generate tanker plans.

ENTRY POINTS: PLANTANK

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, CONTROL, C10, C15, C30, DINDATA, DINDT2, EVENTS, DPENREF, ICLASS, INDATA, IOUT, IRF, KEYS, LASREF, OUTSRT, PRNCON, RECBAS, SNAPON, TANKA, TANKB, TANKER

SUBROUTINES CALLED: CLINDATA, DIRECT, DISTF, DLETE, HDFND, IPUT, ORDER, PRNTAB, RETRV, SLOG, SNAPCON, SNAPIT, STORE, VAM

CALLED BY: PLANOUT

Method:

Enter tanker plan generated consists of seven events: (1) Launch event, (2) Enter Refuel Area, (3) Leave Refuel Area, with (4), (5), (6); and (7) as alternate Recovery events, as shown in table 11.

Tanker plans are generated in the following operation. As each input record is read in, N_t plans are generated: N_a plans for alert tankers, and then $N_t - N_a$ for nonalert tankers.

Each tanker base is first inspected to determine if its tankers are to be automatically allocated. Then, after all bases have been inspected, PLANTANK fills common block /TANKB/ with required data and calls subroutine VAM to allocate those tankers to specific refuel areas in such a way as to minimize the total miles flown by them while servicing all bomber requests.

When VAM has returned its solution, PLANTANK allocates any extra tankers and then proceeds to calculate the time schedules for individual flights.

In the second-strike case, all tankers are sent to their assigned refuel areas at the earliest possible moment, considering delays before launch due to alert or nonalert status as well as the travel time required between base and refuel area.

In the first-strike case, however, they are scheduled as follows. Bombers have been scheduled by PLNTPLAN to arrive at specific refuel areas over a period of time (which may be several hours) so as to satisfy the requirements associated with the CORBOMB input parameter. These bomber refuels have been posted in the matrix IARVLS/ARVLS (I,J) where J indicates data for the Jth refuel to be scheduled by PLNTPLAN,

* First subroutine of overlay TANK.

I=1 contains the scheduled time of the Jth refuel, I=2 contains the assigned refuel area.

As each tanker is processed, the IARVLS array is searched for the first unserviced bomber refuel which is to occur at the refuel area to which the tanker has been assigned by subroutine VAM. When found, the bomber time of arrival is retrieved, the tanker is scheduled to launch so as to arrive at the refuel area .1 hour prior to the bomber, and the IARVLS entry is set to zero. to indicate that the bomber has been serviced. If the search finds no unserviced bomber at the refuel area, the tanker is extra, thus PLANTANK schedules it to arrive .1 hour before the earliest bomber at the area (stored in array ARTIME).

After scheduling has been completed, distances from refuel area to recovery bases are calculated for each tanker, the recovery events are ordered by ascending distance, and EVENTAPE, PLANTAPE, and printed reports are output with tanker plans according to user options.

Figure 111 illustrates subroutine PLANTANK.

Table 11. Tanker Plan

<u>Event Type</u>	<u>Time</u>	<u>Place</u>
Launch	Delay	INDEXTK
Enter Refuel Area	$DIST/V_t$	IREFTK as set by PLANTANK
Leave Refuel Area	TTOS	IREFTK as set by PLANTANK
Recover ₁	DI_1/V	(RCBLAT, RCBLONG) ₁
Recover ₂	DI_2/V_t	(RCBLAT, RCBLONG) ₂
Recover ₃	DI_3/V_t	(RCBLAT, RCBLONG) ₃
Recover ₄	DI_4/V_t	(RCBLAT, RCBLONG) ₄

Where DIST = Distance from tanker base to refuel area
 DI_x = Distance from refuel area to recovery base_x

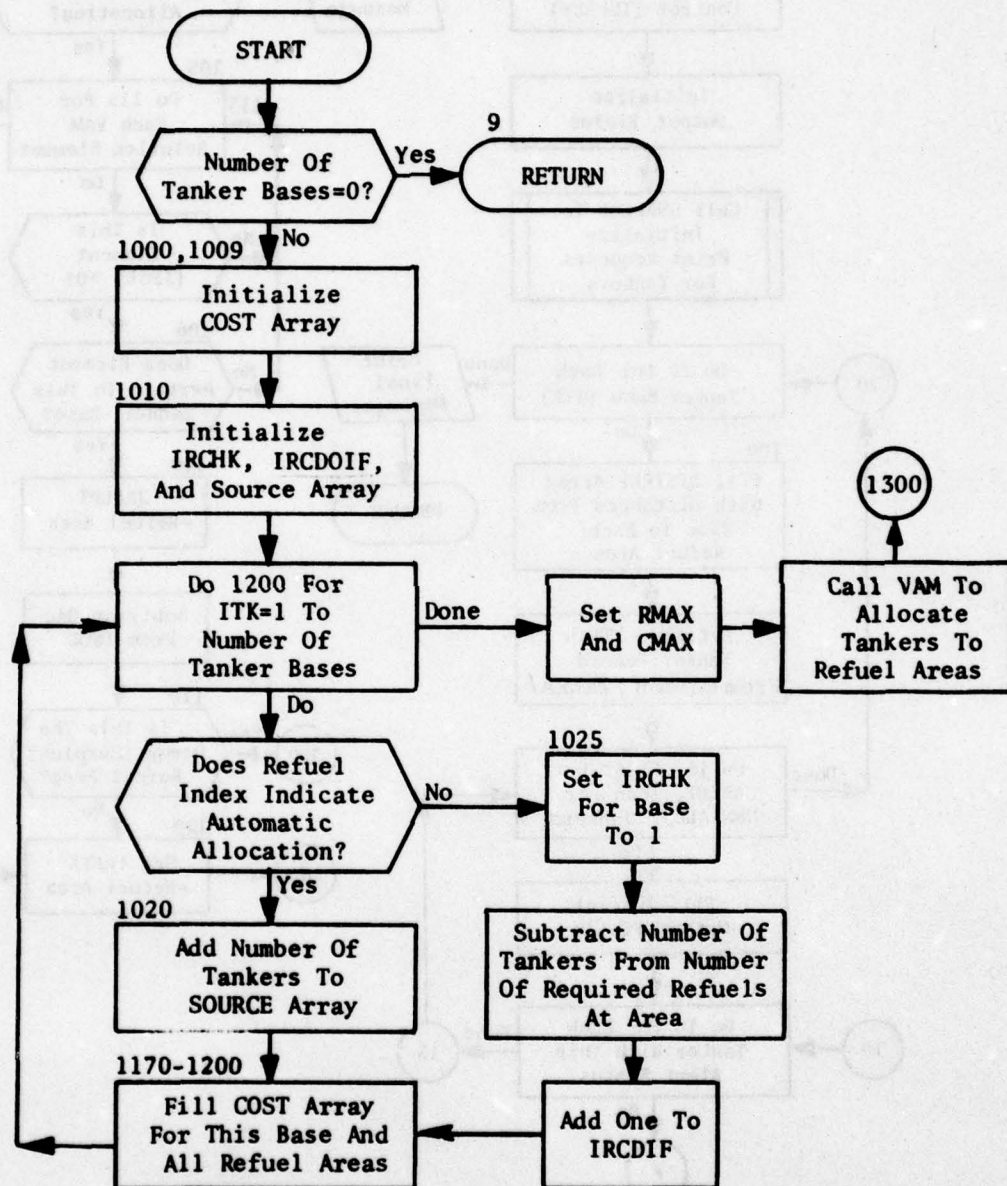


Figure 111. Subroutine PLANTANK (Part 1 of 4)

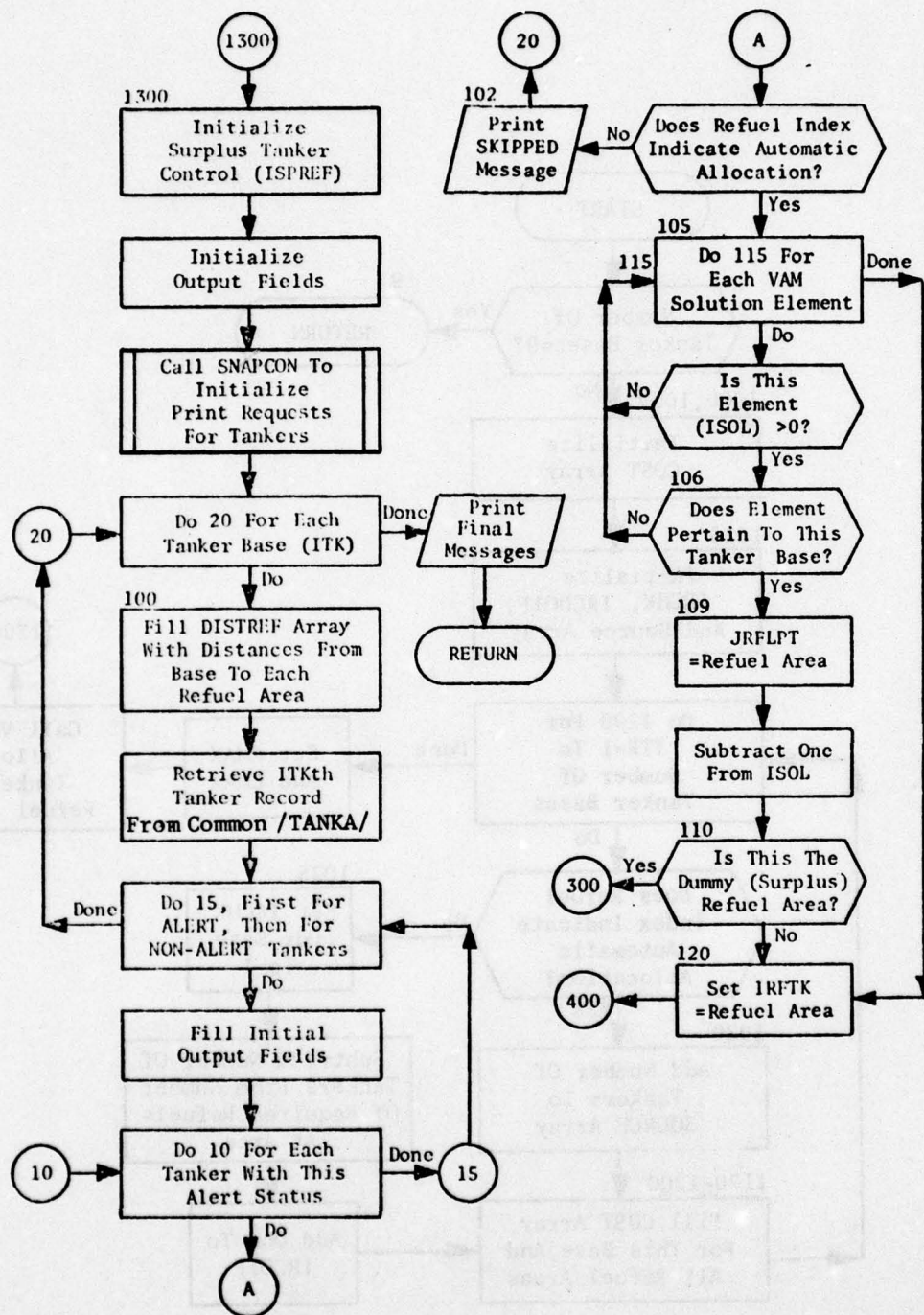


Figure 111. (Part 2 of 4)

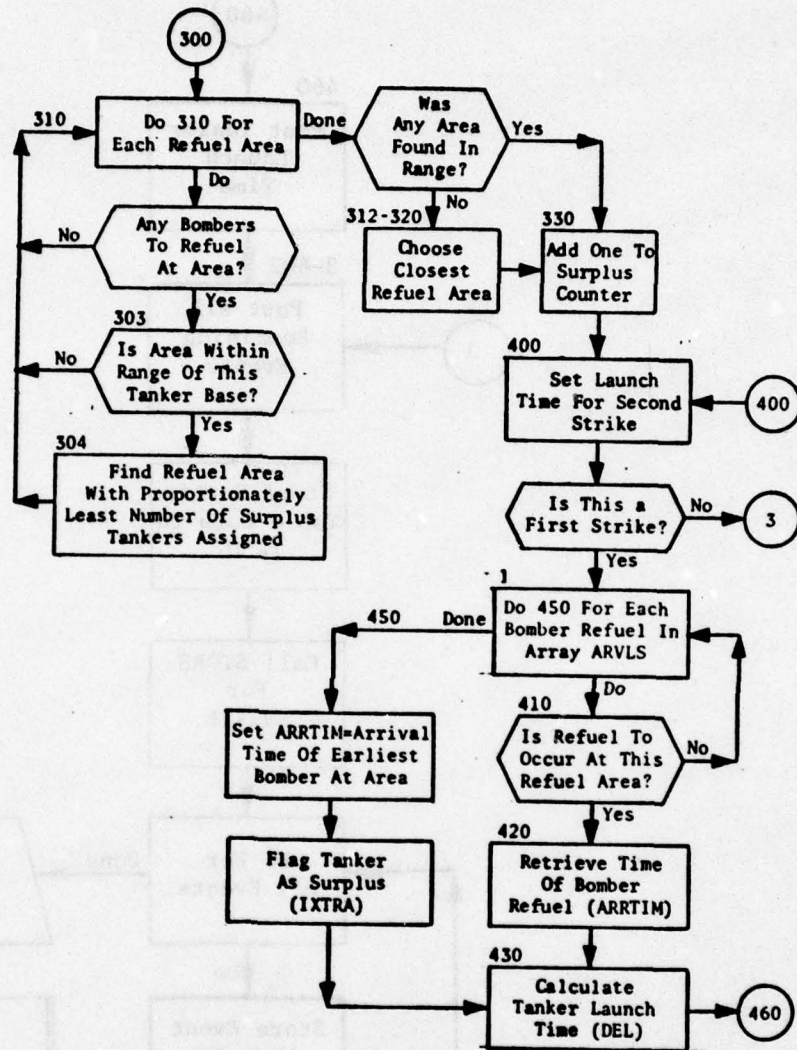


Figure 111. (Part 3 of 4)

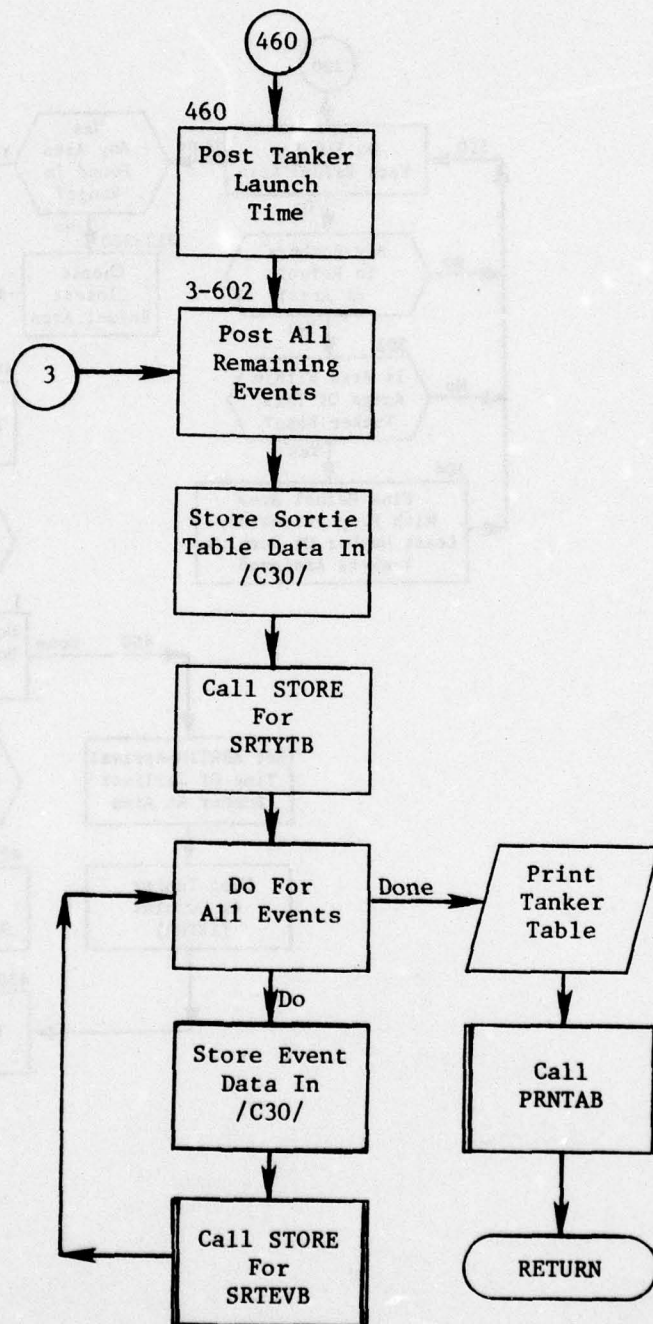


Figure 111. (Part 4 of 4)

4.9.1 Subroutine PRNTAB

PURPOSE: To print the final tanker allocation tables.

ENTRY POINTS: PRNTAB

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, C30, DPENREF, IRF

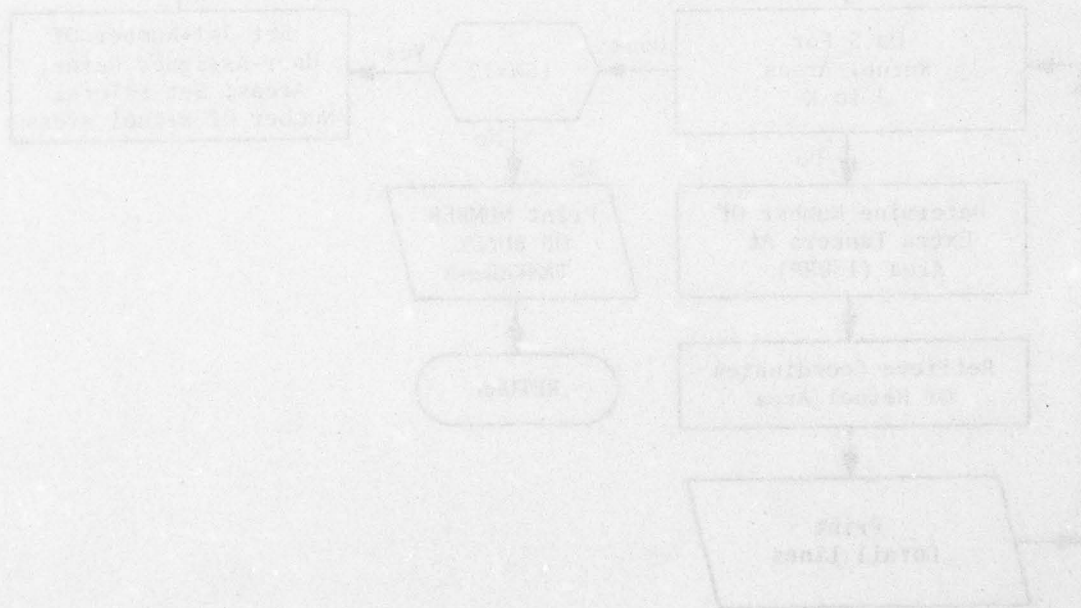
SUBROUTINES CALLED: CONVLL

CALLED BY: PLNTPLAN

Method:

Subroutine PRNTAB outputs tanker allocation information, first for the user-assigned refuel areas, then, after skipping a line, for the refuel areas calculated by PLNTPLAN.

Subroutine PRNTAB is illustrated in figure 112.



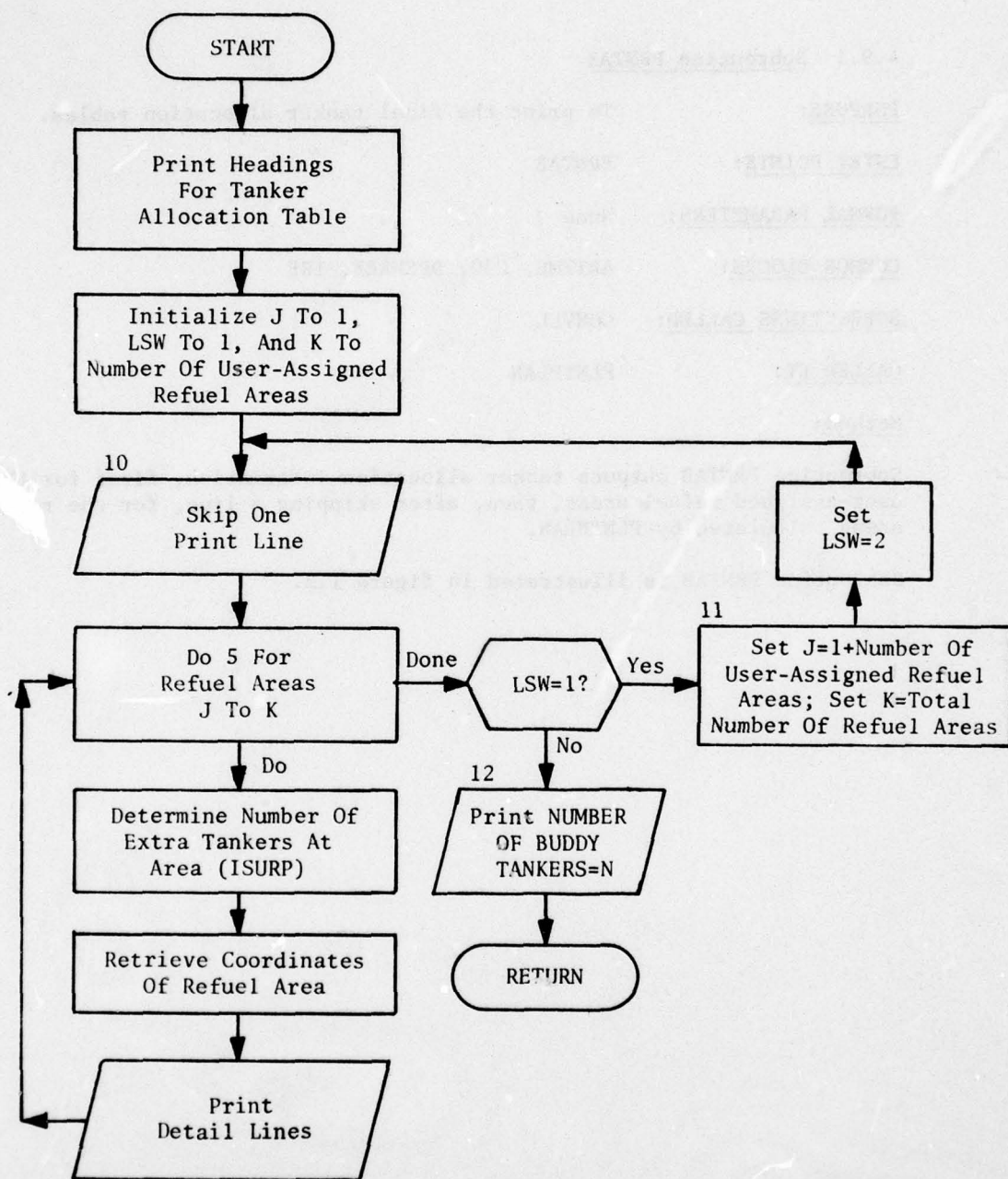


Figure 112. Subroutine PRNTAB

4.9.2 Subroutine VAM

PURPOSE: To solve the transportation problem of allocating available tankers to bomber refuels so as to minimize the total tanker miles flown.

ENTRY POINTS: VAM

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, TANKB

SUBROUTINES CALLED: GLOG, SLOG

CALLED BY: PLANTANK

Method:

The task of allocating tankers to refuel areas in such a way as to service all bomber refuel requirements is considered by subroutine PLANTANK to be a form of the classic transportation problem. Hence, it structures the data as such in common /TANKB/, and calls subroutine VAM to apply VOGEL's Approximation Method to obtain a solution.

A general statement of the problem described in the following involves the variables shown in figure 113.

Each cell in figure 113 has two entries associated with it:

- a. $COST(i,j)$ = distance from base i to refuel area j + safety factor of .5 miles.
- b. $X(i,j)$ = number of tankers at base i to be assigned to refuel area j .

The cost matrix is an input to the algorithm; the X matrix is its solution.

The statement of the transportation problem to be solved is:

Given: all i, j, a_i, b_j , and $COST(i,j)$,

Find: all $X(i,j)$ such that the total number of tanker miles flown

$$\left(\sum_{i=1}^R \sum_{j=1}^C [COST(i,j) * X(i,j)] \right)$$

		j = Refuel area number					
j =		1	2	3	.	.	.
i = Tanker base number	i = 1	*	**				a_1
	2						a_2
	3						a_3
	.						.
	R						a_R
		b_i	b_2	b_3			b_c
		b _j = Total number of tankers required at refuel area j					
		a _i = Total number of tankers available at tanker base i					

* $\frac{\text{COST}(1,1)}{X(1,1)}$ ** $\frac{\text{COST}(1,2)}{X(1,2)}$

Figure 113. Base/Refuel Area Sample Matrix

is minimized, subject to the constraints that

- a. the total number of tankers assigned from base i must equal the total number of tankers available at base i

$$\left(\sum_{j=1}^C X(i,j) = a_i \text{ for } 1 \leq i \leq R \right)$$

- b. the total number of tankers assigned to refuel area j must equal the total number required at refuel area j

$$\left(\sum_{i=1}^R X(i,j) = b_j \text{ for } 1 \leq j \leq C \right)$$

A dummy refuel area is created to handle extra tankers, which are later reassigned by subroutine PLANTANK.

The FORTRAN labels used in VAM rather than the symbols above are:

<u>Table Symbol</u>	<u>FORTTRAN Name</u>
C	CMAX
R	RMAX
a	SOURCE(I)
b	SINK(J)
COST(i,j)	COST(I,J)
X(i,j)	ISOL(K), where X(RBASLOC(I),CBASLOC(J))=ISOL(K)

The solution is found using Vogel's Approximation Method, a standard operations research technique. The steps of the procedure are:

- For each row and column in the COST matrix, calculate the difference between the smallest and next-smallest entry (row and column penalties).
- Select the row or column with the largest difference.
- Allocate as many tankers as possible to the smallest COST cell in that row or column.
- Allocate zero elsewhere in the row or column where the supply (tankers) or demand (refuel requests) has been exhausted.
- Make the only feasible allocation in any rows or columns having only one cell without an allocation of tankers.
- Eliminate all fully allocated rows and columns from further consideration. Stop if no rows or columns remain. Otherwise,
- Begin again, using the modified COST matrix.

A nondegenerate basic feasible solution will have $(C_{MAX}+R_{MAX}-1)$ non-zero allocations in the ISOL array.

Subroutine VAM is illustrated in figure 114.

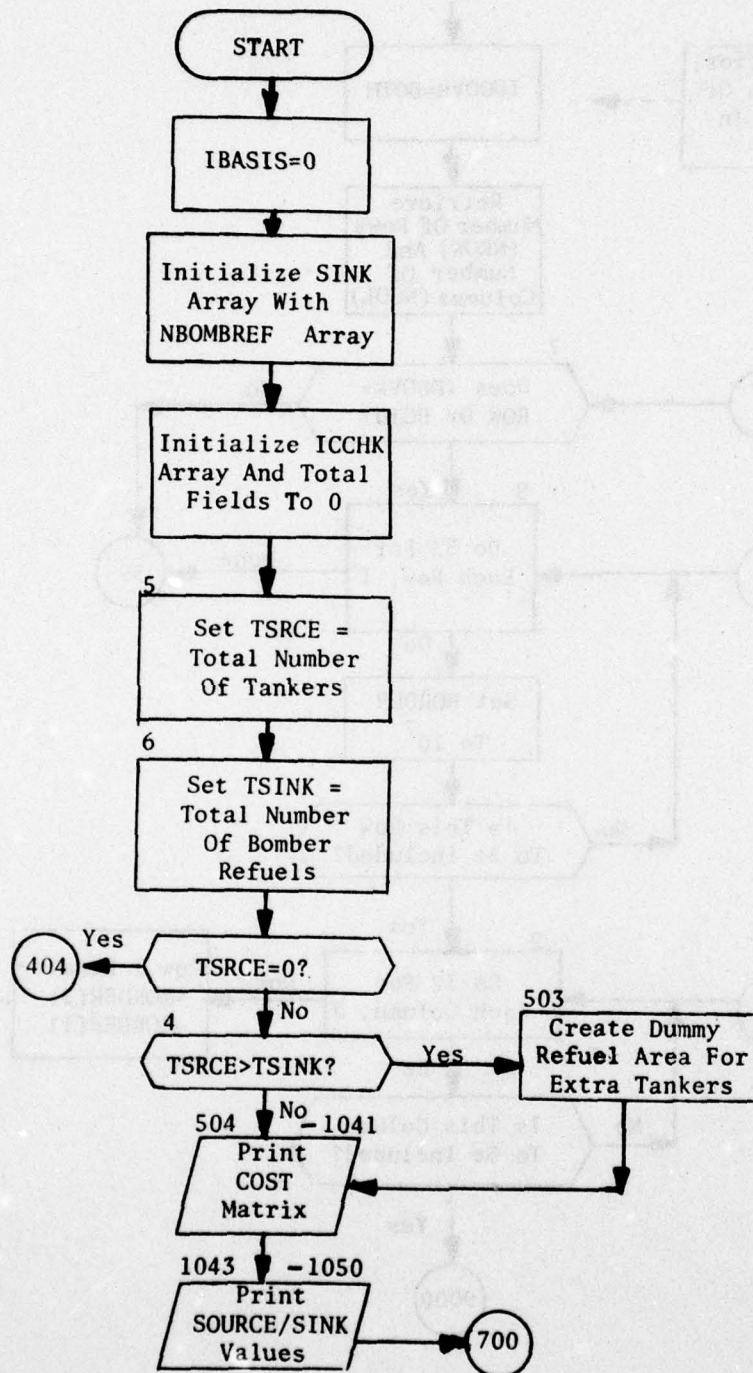


Figure 114. Subroutine VAM (Part 1 of 8)

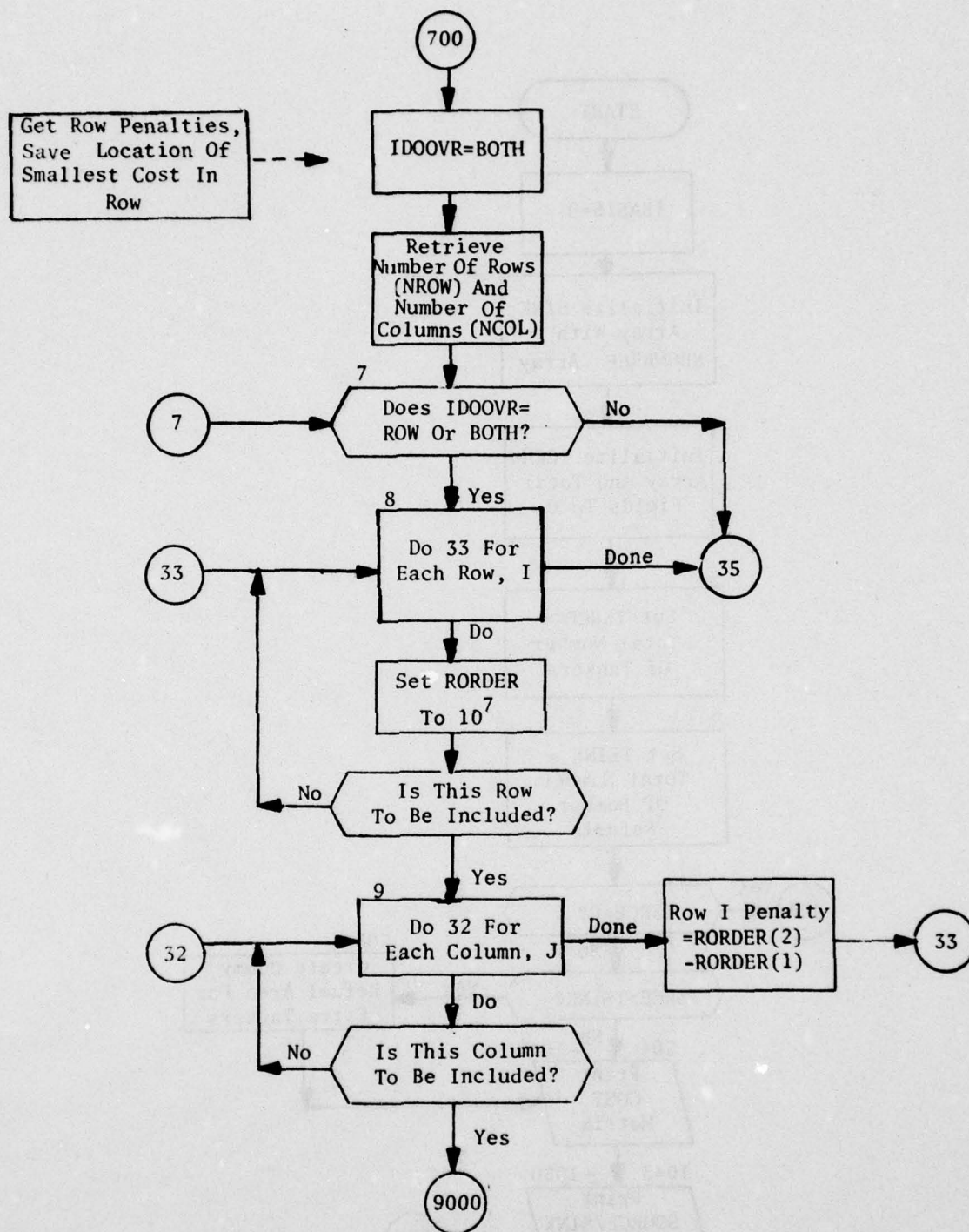


Figure 114. (Part 2 of 8)

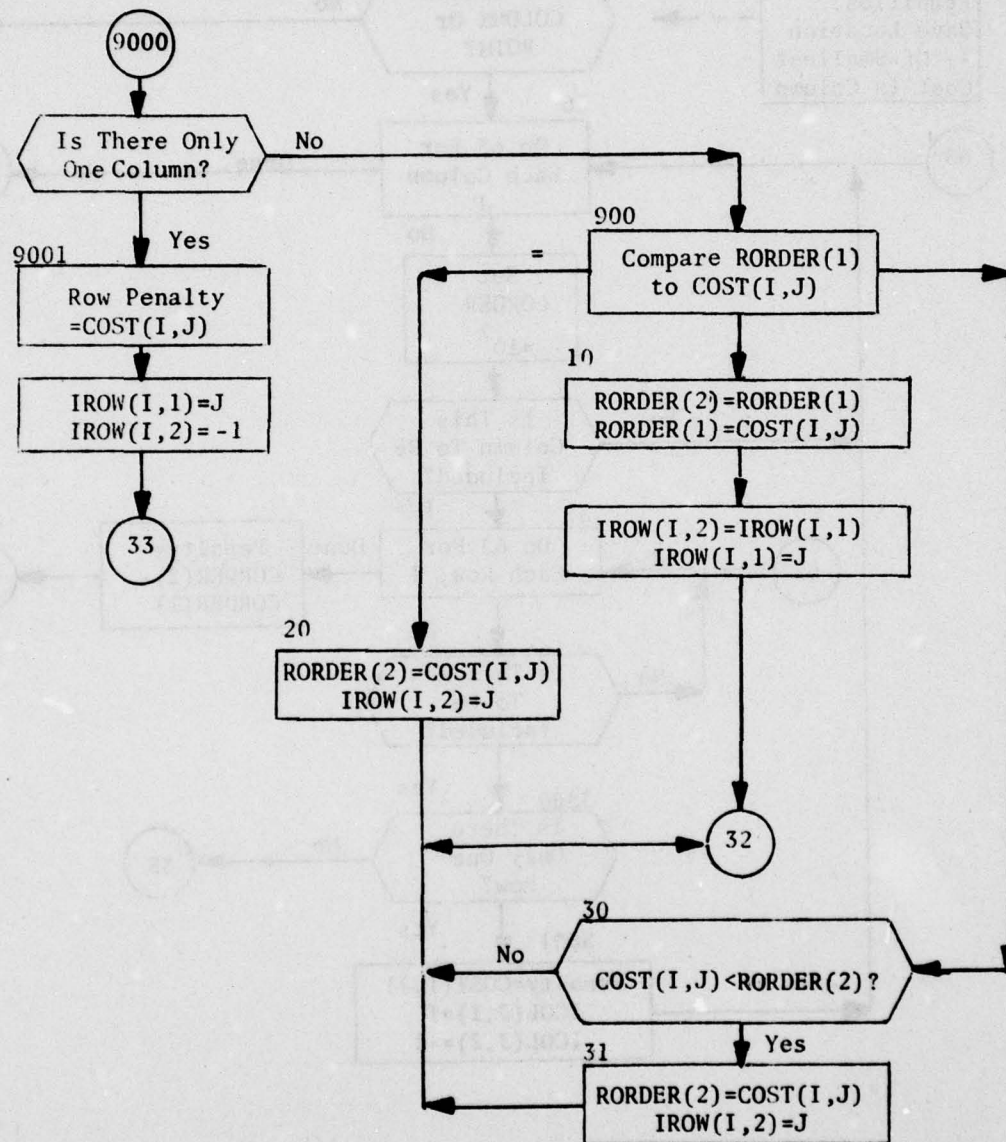


Figure 114. (Part 3 of 8)

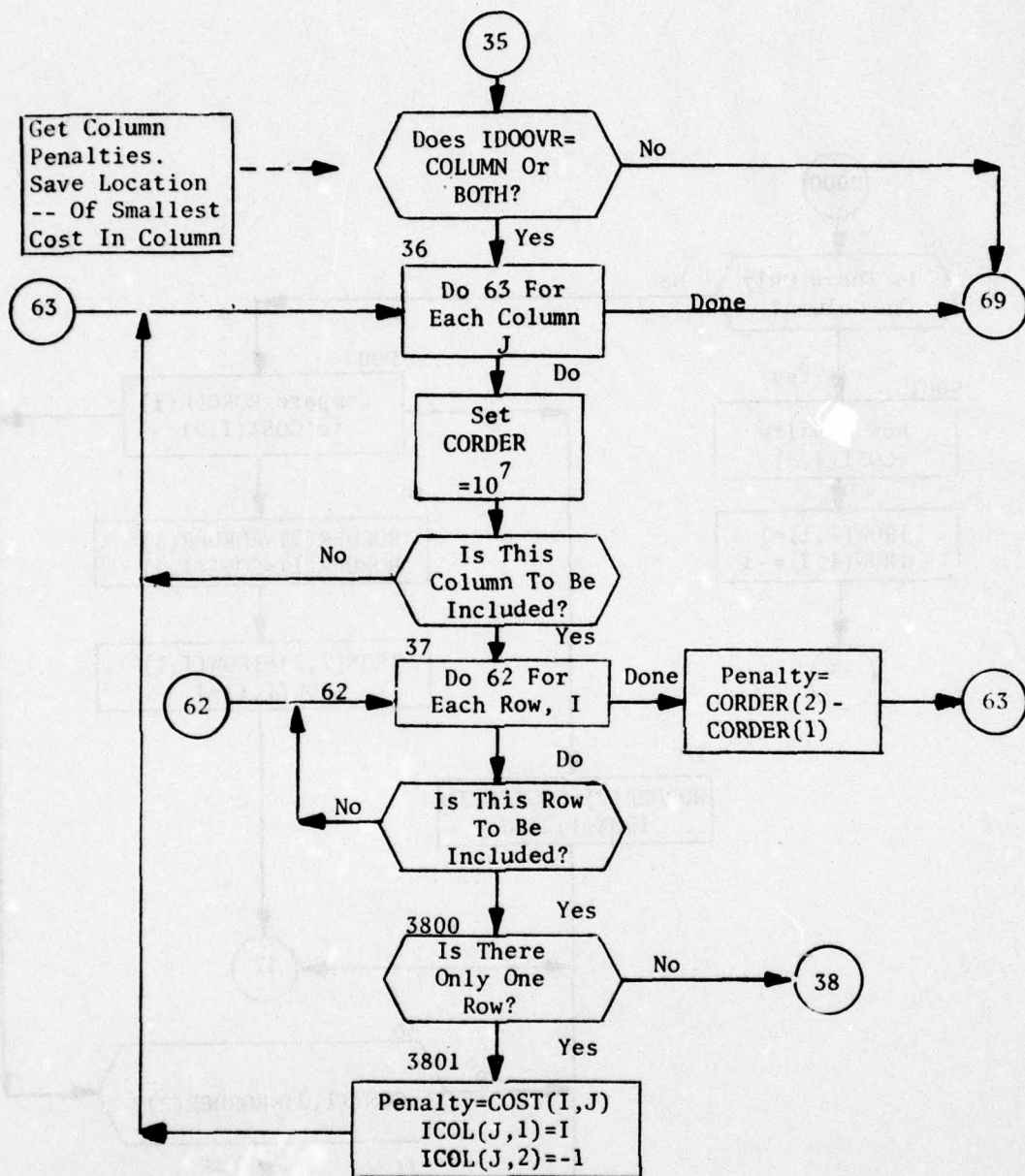


Figure 114. (Part 4 of 8)

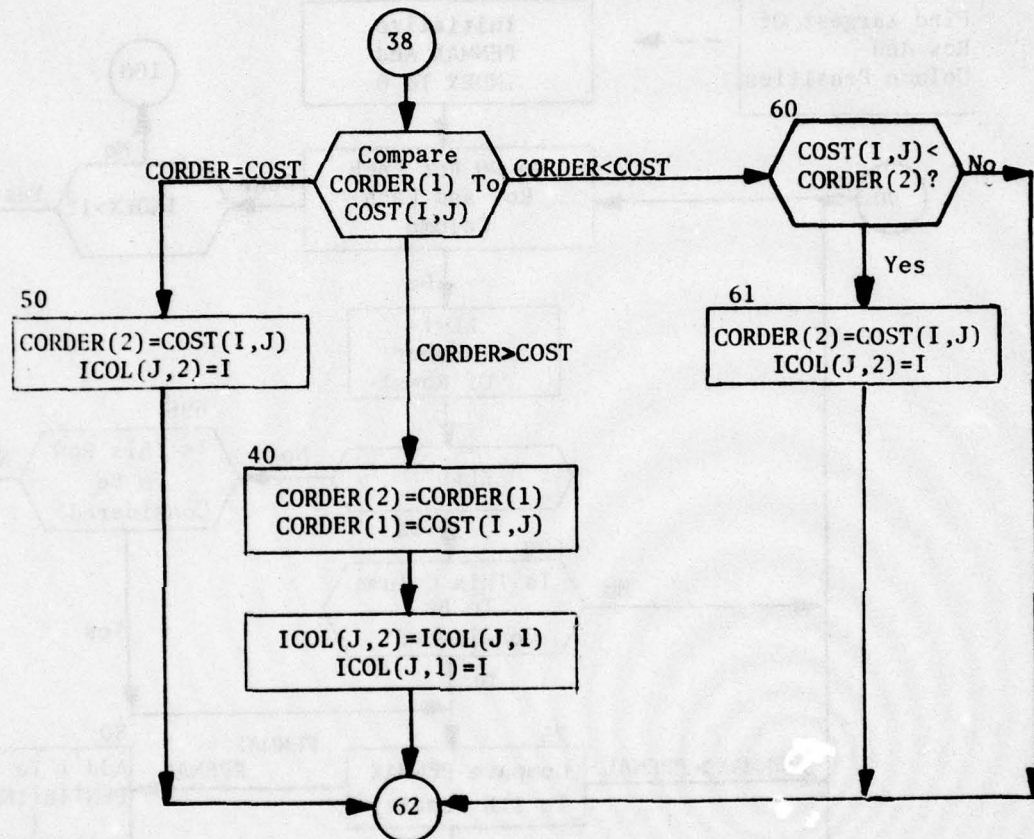


Figure 114. (Part 5 of 8)

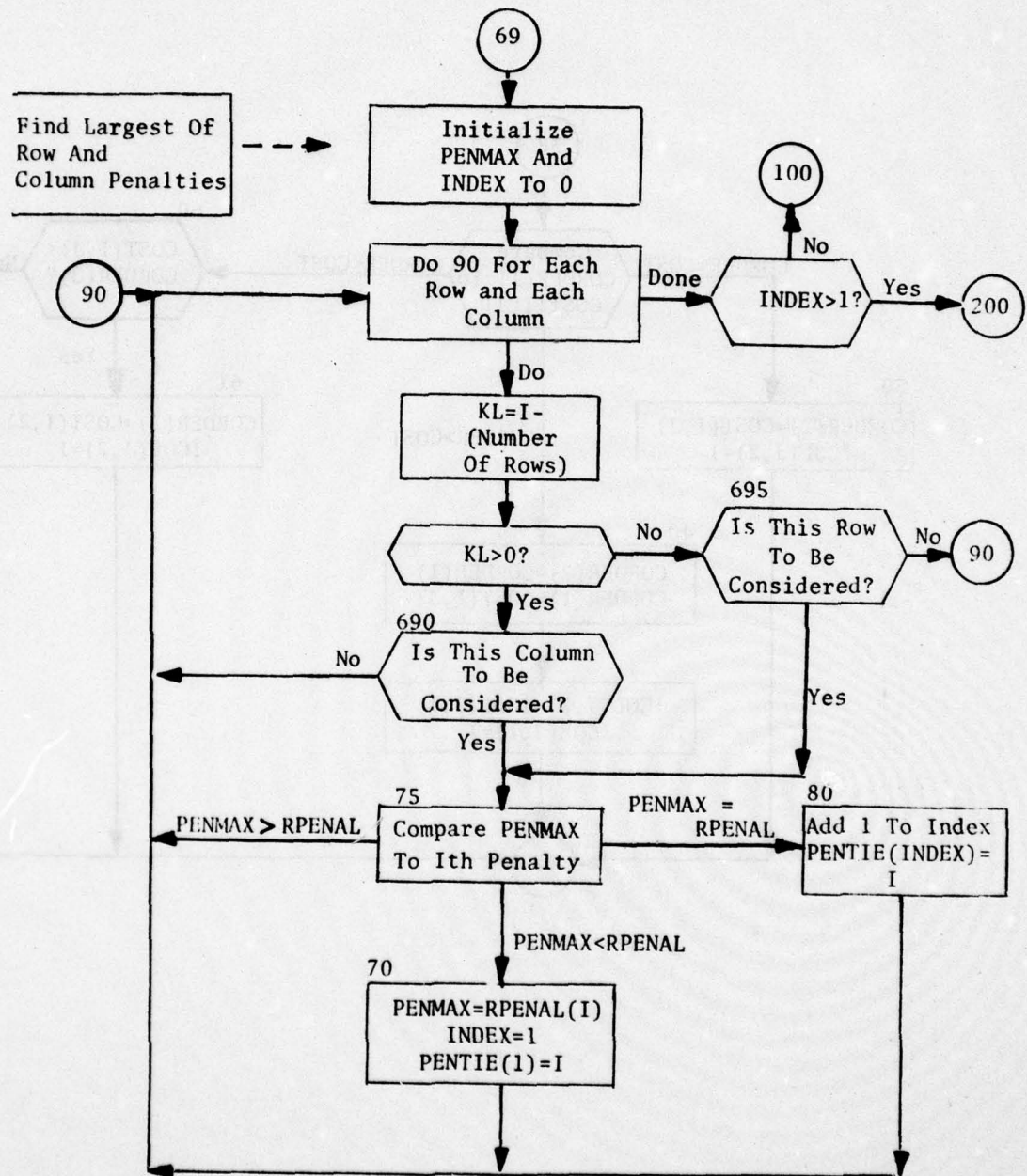
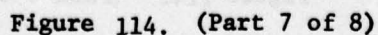


Figure 114. (Part 6 of 8)



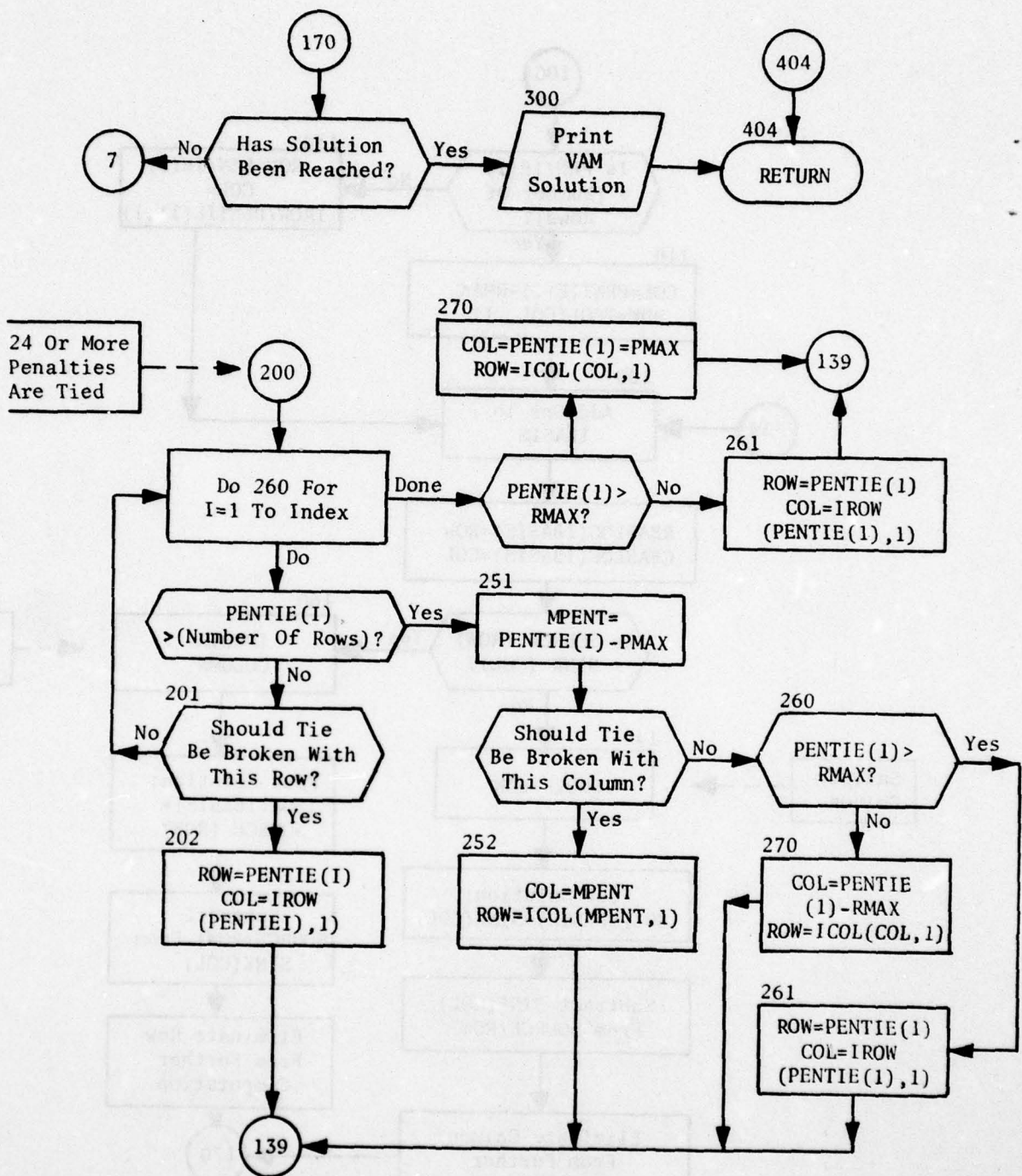


Figure 114. (Part 8 of 8)

4.10 Subroutine INTRFACE*

PURPOSE: Driver subroutine for overlay which creates ABTAPE and STRIKE tapes

ENTRY POINTS: INTRFACE

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, C10, C15, C30, EVCOM, NOFSYS, OOPS, PLTYP, PRNCON, PSW

SUBROUTINES CALLED: ABOUT, DIRECT, HDFND, HEAD, IFSET, NEXTTT, RDCLAUSE, RETRV, STOUT, TYPFIND, WEPDATA

CALLED BY: ENTMOD (PLANOUT)

Method:

First the print switches in block /PSW/ are set according to block /PRNCON/. Also if the frequency for the ABTAPE print option (13) is not 1, the print report code (ABUNIT) is reset to 42. RDCLAUSE and IFSET are now called to read the GAMETIME, FUNCOM and all IF and SETTING clauses. WEPDATA is now called for weapon data.

The sortie header is retrieved and each sortie processed in order. After the last sortie any active tapes have file marks written on them and are rewound. For each sortie the following procedure is followed: TYPFIND is called to set type names and numbers. If ABTAPE is active, ABOUT is called to produce an A-Record.

Then each event of the sortie is retrieved. For each event the data in block /EVCOM/ is updated. If the event is a weapon assignment event and STRIKE tape is active, STOUT is called. For any event if ABTAPE is active ABOUT is called for a B-Record.

Subroutine INTRFACE is illustrated in figure 115.

* First subroutine of overlay INTR.

AD-A058 406

COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK). VOLU--ETC(U)
APR 78 D J SANDERS, P F MAYKRANTZ, J M HERRON

F/G 15/7

UNCLASSIFIED

CCTC-CSM-MM-9-77-VOL-4-PT SBIE-AD-E100 085

NL

4 OF 5
ADA
058406



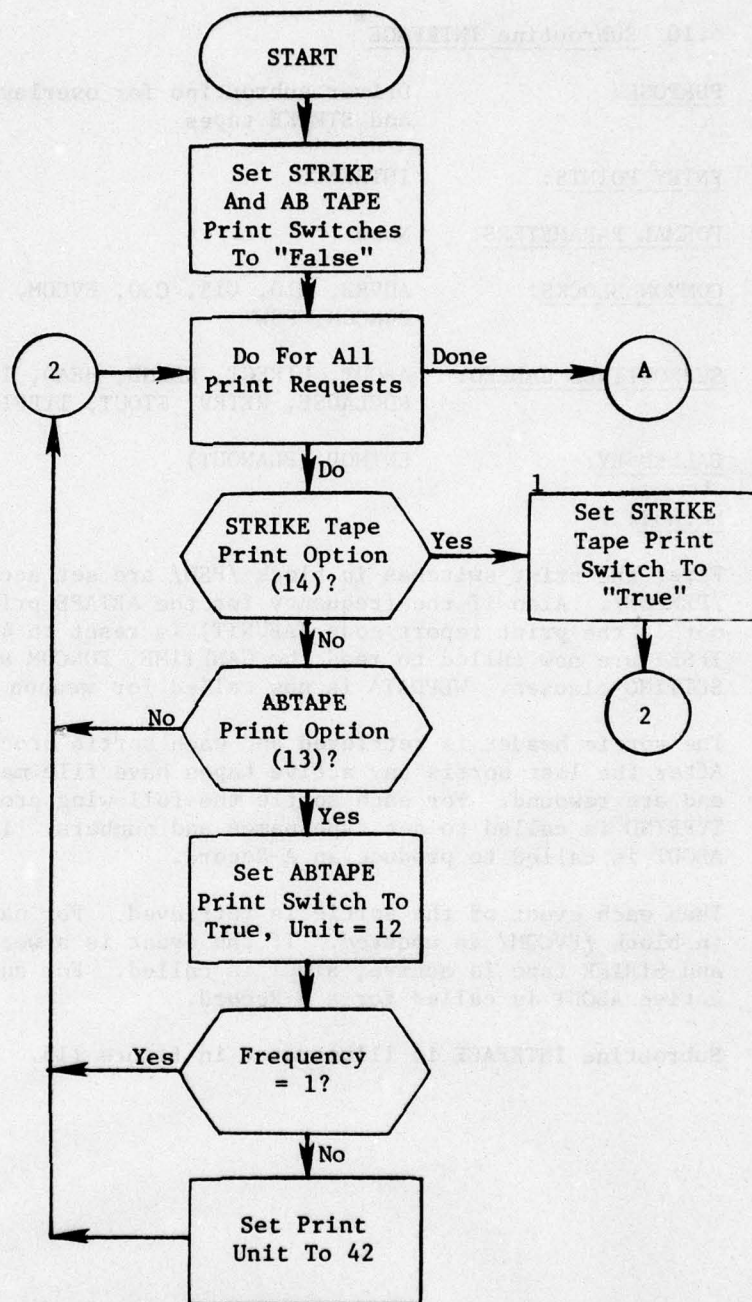


Figure 115. Subroutine INTRFACE (Part 1 of 5)

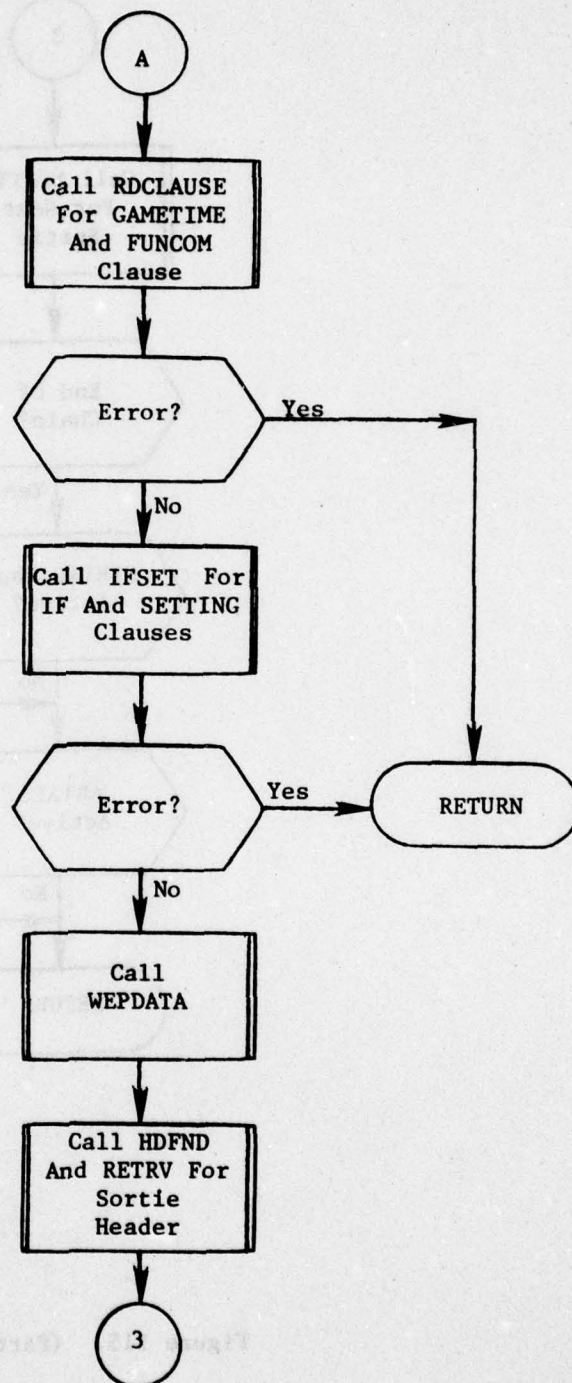


Figure 115. (Part 2 of 5)

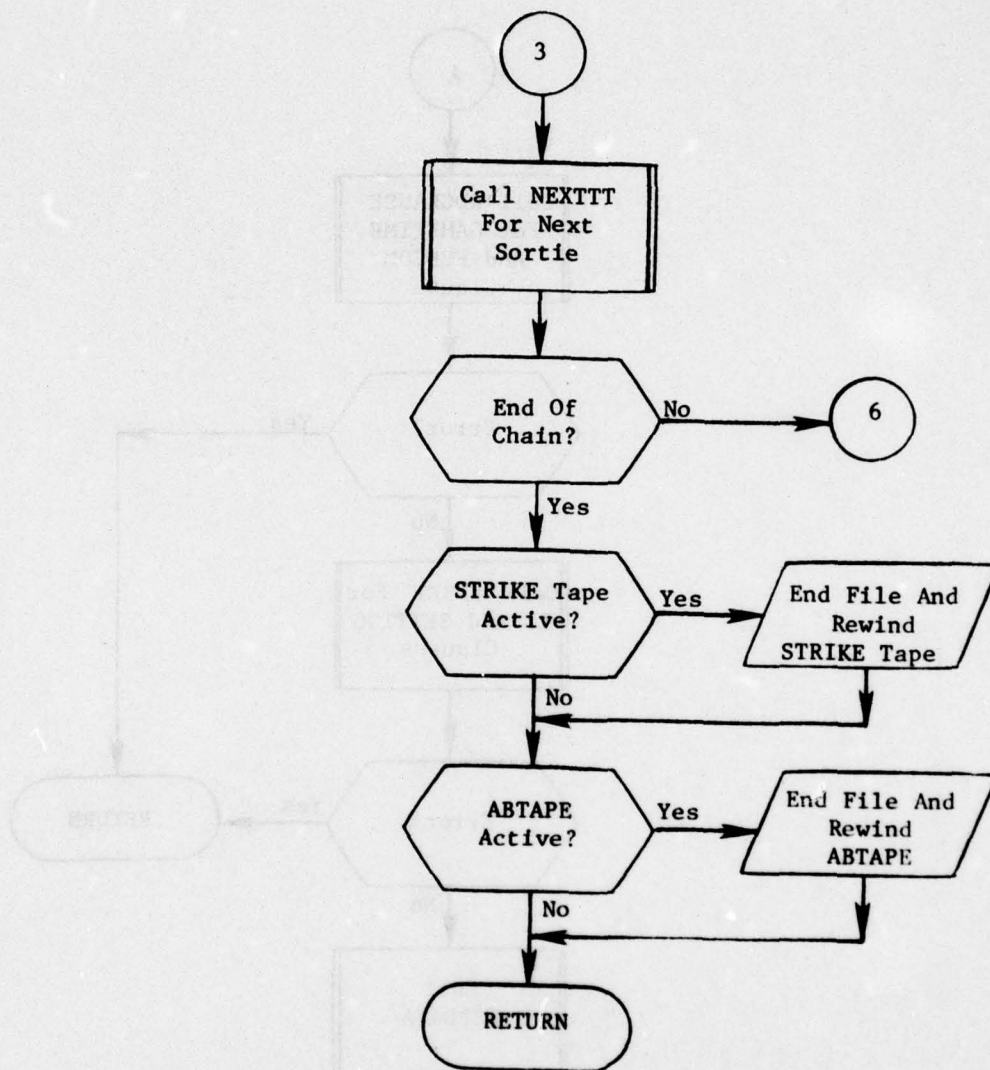


Figure 115. (Part 3 of 5)

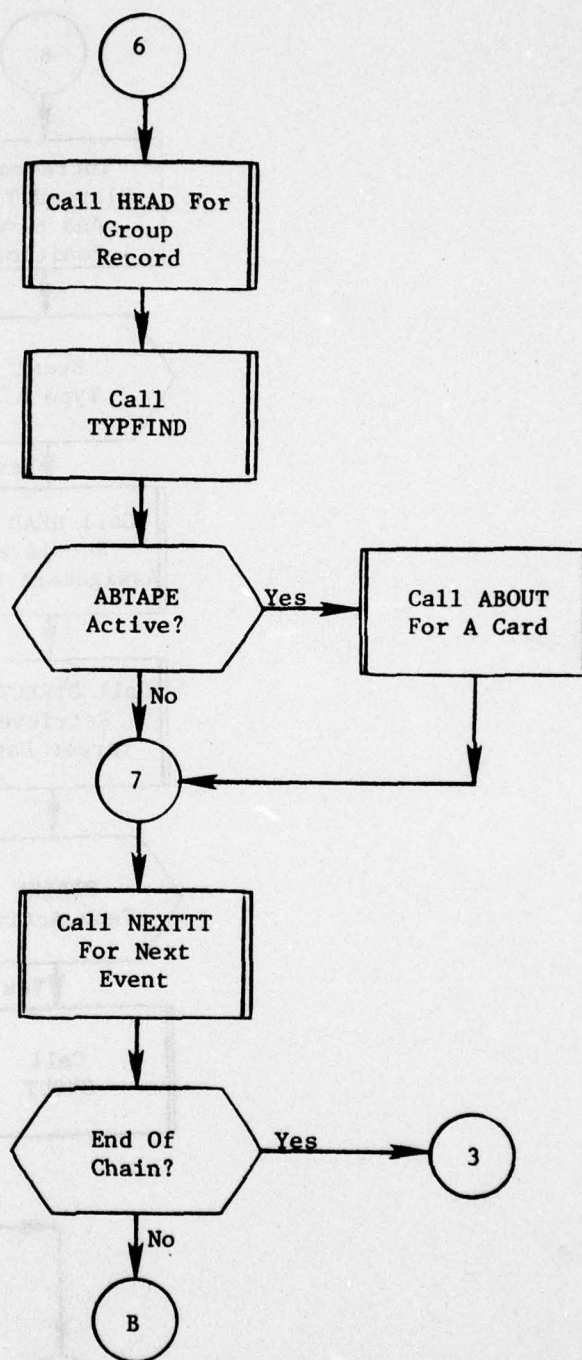


Figure 115. (Part 4 of 5)

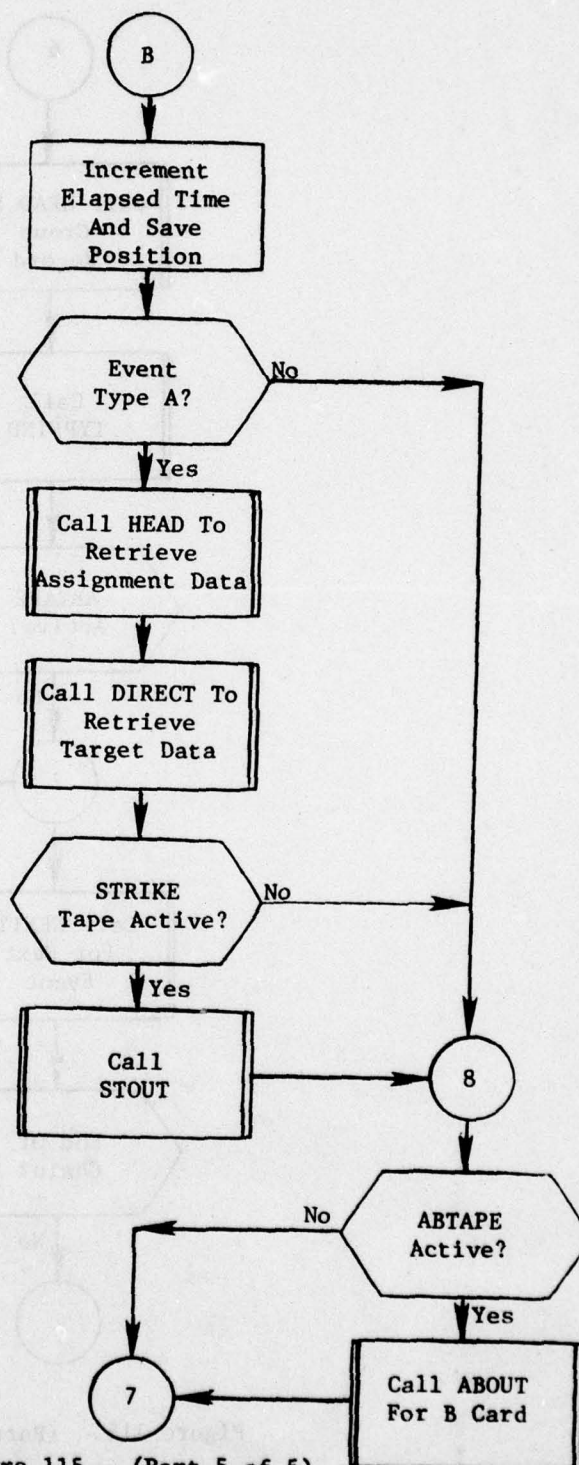


Figure 115. (Part 5 of 5)

4.10.1 Subroutine ABOUT

PURPOSE: Write records on ABTAPE.

ENTRY POINTS: ABOUT

FORMAL PARAMETERS: IAB - =1: Produce A-Record
=2: Produce B-Record

COMMON BLOCKS: C30, DEFVAR, EVCOM, GRPSTF, IFSCOM, MODE, PAYSTF, PLTYP, PSW, TYPSTF, WHDSTF

SUBROUTINES CALLED: CONVLL, FINDTIME, IAZIM, IFUNCT, IGETHOB, INFORM, NOP, NTIME, XSET, XWHERE

CALLED BY: INTRFACE

Method:

First data is stored in variable block /DEFVAR/. Depending on whether the call is for an A or B record, different values are filled. The positions in /DEFVAR/ correspond to record fields as delineated in figure 60. For an A-Record, NTIME is called to set the time of launch. For a B-Record, FINDTIME is called and IAZIM is used for missile azimuth and back-azimuth.

When all data is stored, all pairs of IF and SETTING clauses are executed. Finally, the desired record is formatted using INFORM and written on the output unit (LTN 16).

Subroutine ABOUT is illustrated in figure 116.

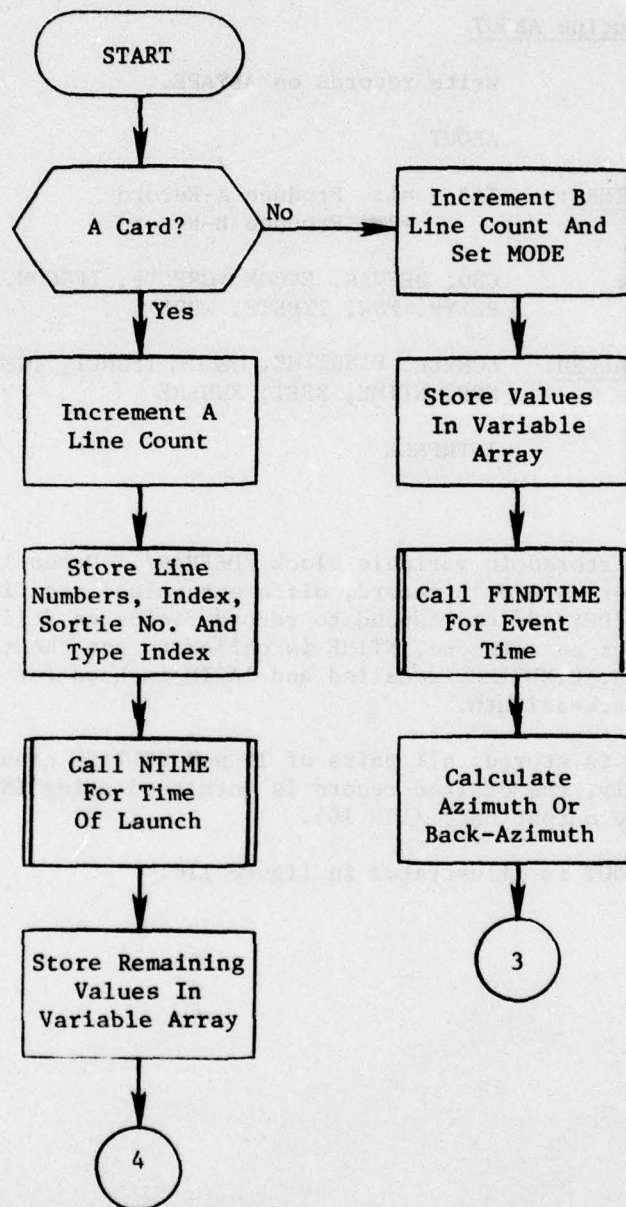


Figure 116. Subroutine ABOUT (Part 1 of 4)

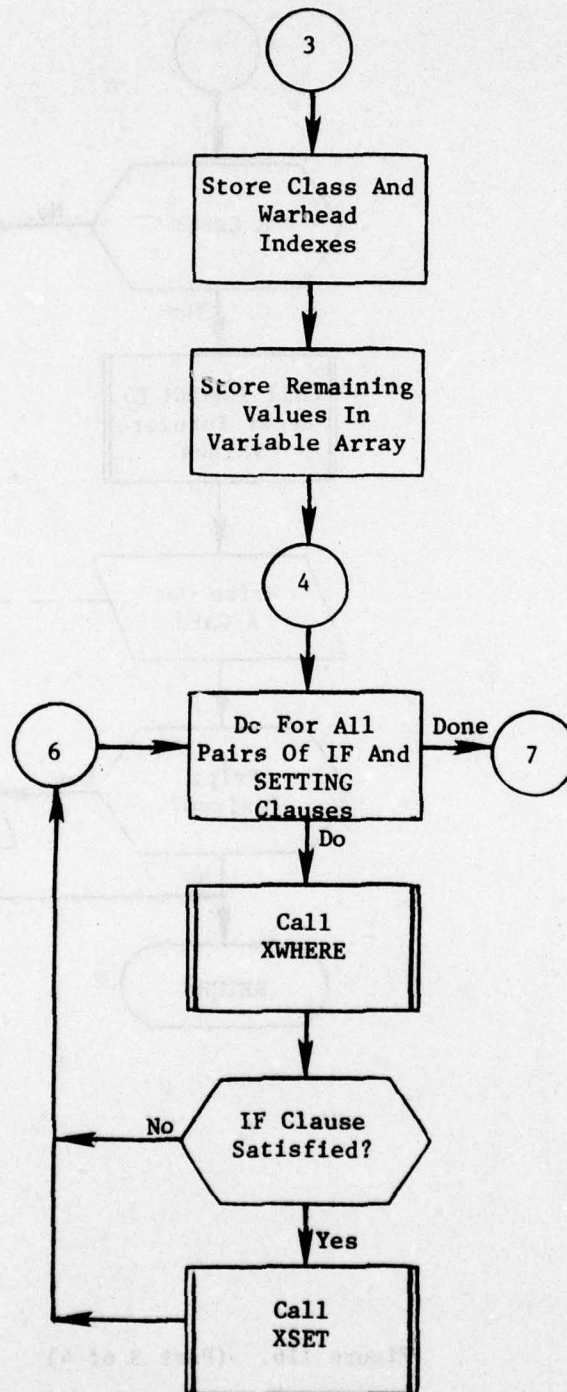


Figure 116. (Part 2 of 4)

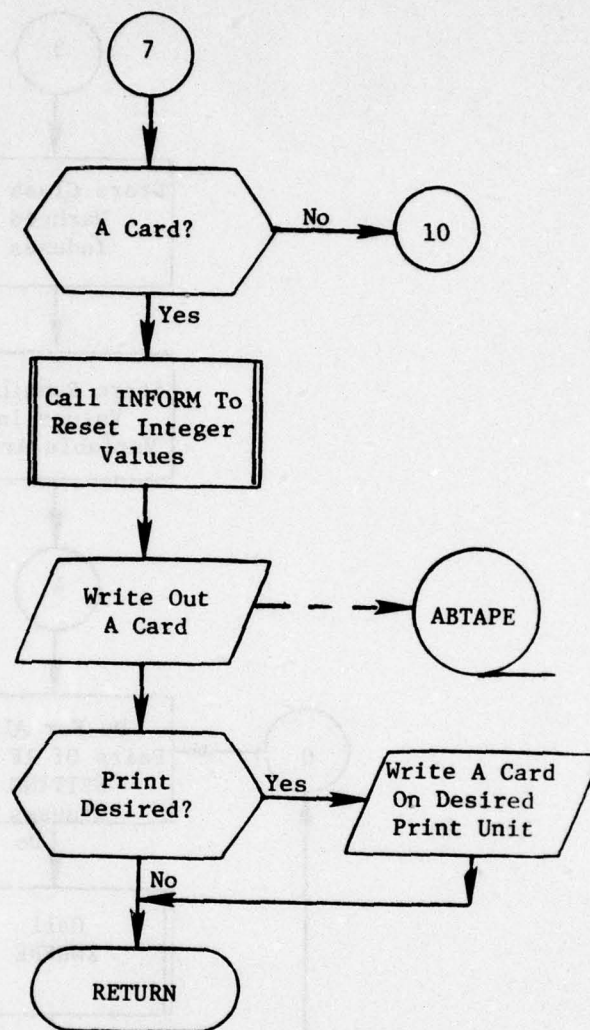


Figure 116. (Part 3 of 4)

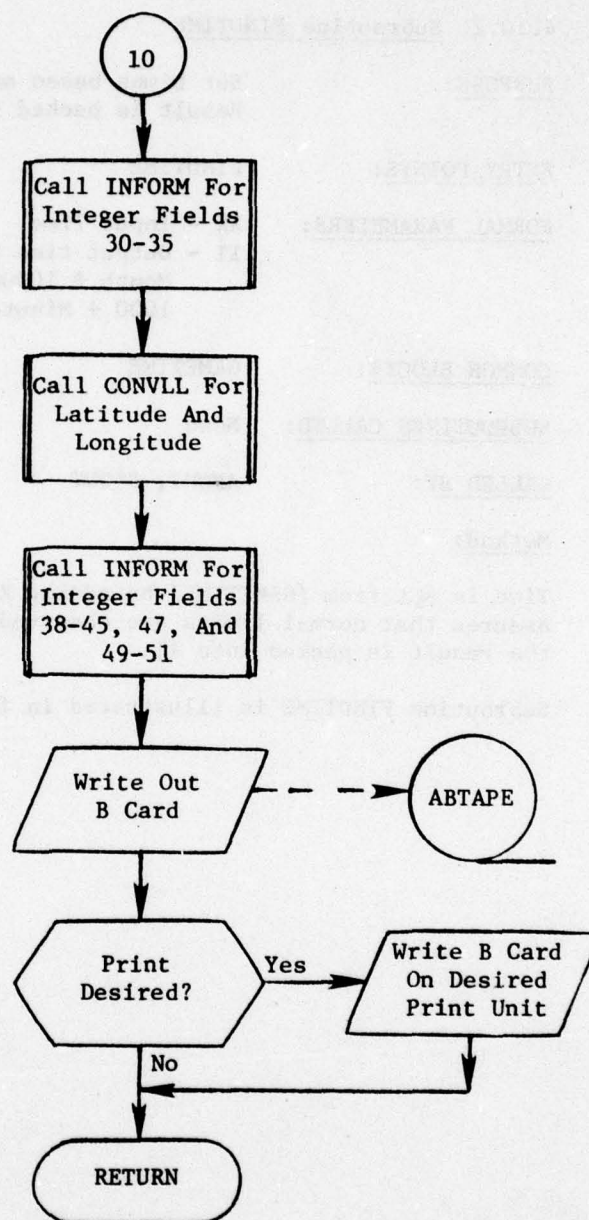


Figure 116. (Part 4 of 4)

4.10.2 Subroutine FINDTIME

PURPOSE: Set times based on input and /GAMETIME/ block.
Result is packed into output parameter

ENTRY POINTS: FINDTIME

FORMAL PARAMETERS: XX - Input Time
II - Output time packed as follows
 $\text{Month} * 100000000 + \text{Day} * 100000 + \text{Hour} * 1000 + \text{Minute} * 100 + \text{Second}$

COMMON BLOCKS: GAMETIME

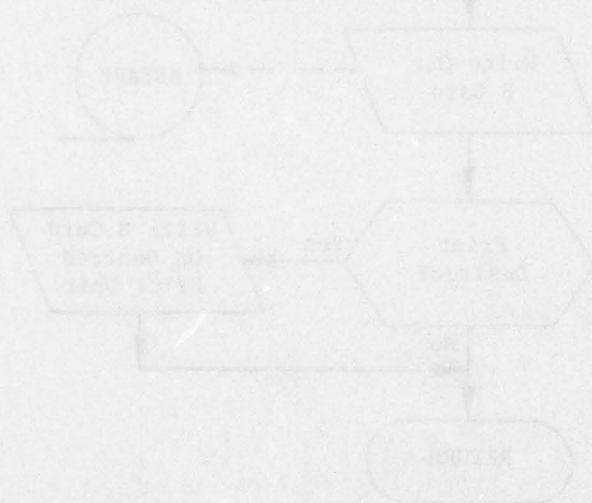
SUBROUTINES CALLED: None

CALLED BY: ABOUT, STOUT

Method:

Time is set from /GAMETIME/ by adding XX to HHR. Then the subroutine assures that normal limits are observed (i.e., 24 hours/day, etc.) and the result is packed into II.

Subroutine FINDTIME is illustrated in figure 117.



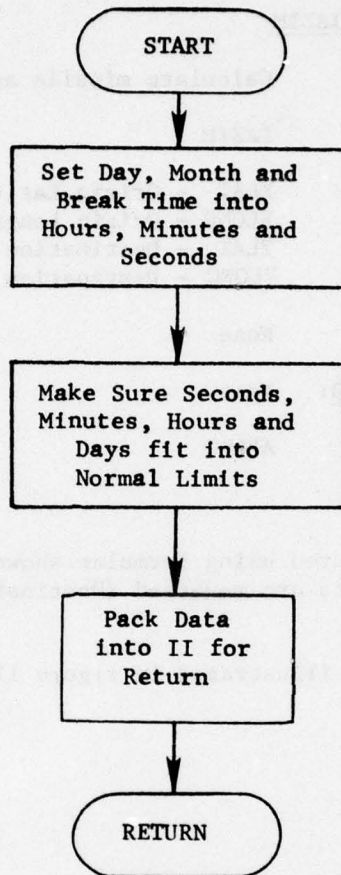


Figure 117. Subroutine FINDTIME

4.10.3 Function IAZIM

PURPOSE: Calculate missile azimuths and back-azimuths.

ENTRY POINTS: IAZIM

FORMAL PARAMETERS: XLAT - Origin Latitude
XLONG - Origin Longitude
YLAT - Destination Latitude
YLONG - Destination Longitude

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: ABOUT

Method:

Azimuth is calculated using formulae shown in figure 118. To calculate back-azimuth inputs are reversed (Destination = XLAT, XLONG; Origin = YLAT, YLONG).

Function IAZIM is illustrated in figure 118.

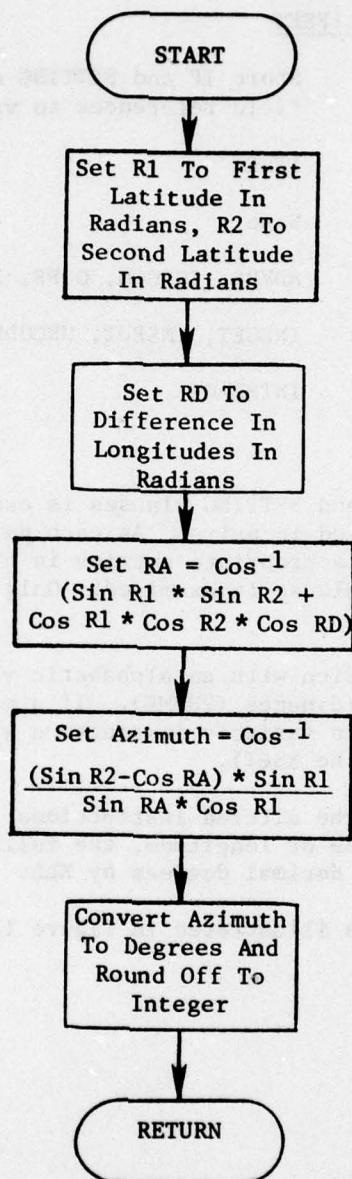


Figure 118. Function IAZIM

4.10.4 Subroutine IFSET

PURPOSE: Store IF and SETTING clause indexes and change field references to variables

ENTRY POINTS: IFSET

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, IFSCOM, OOPS, ZEES

SUBROUTINES CALLED: INSGET, INSPUT, UNCODE, XLL

CALLED BY: INTRFACE

Method:

Processing for IF and SETTING clauses is essentially the same. The clauses are processed in pairs. As each pair is processed, their indexes are saved in appropriate entries in block /IFSCOM/. Each instruction of each clause is examined. Only two sorts of instructions are altered.

First, any instruction with an alphabetic value has that value compared to the list of field names (VNAME). If a match is found the instruction is altered to a variable instruction with an index to the /DEFVAR/ block (see subroutine XSET).

Second, if one of the altered instructions refer to a field whose value is either a latitude or longitude, the following instruction has its value converted to decimal degrees by XLL.

Subroutine IFSET is illustrated in figure 119.

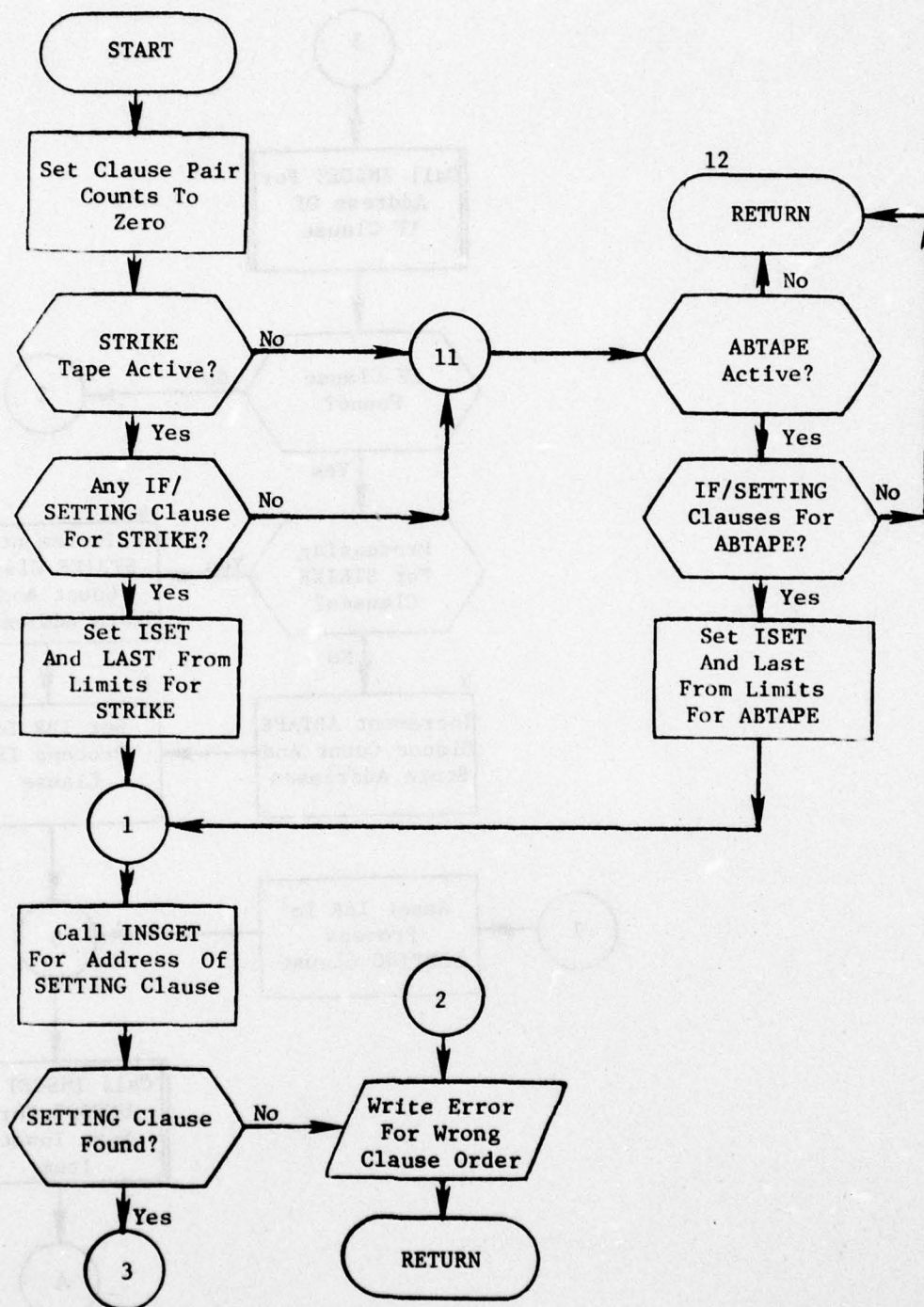


Figure 119. Subroutine IFSET (Part 1 of 4)

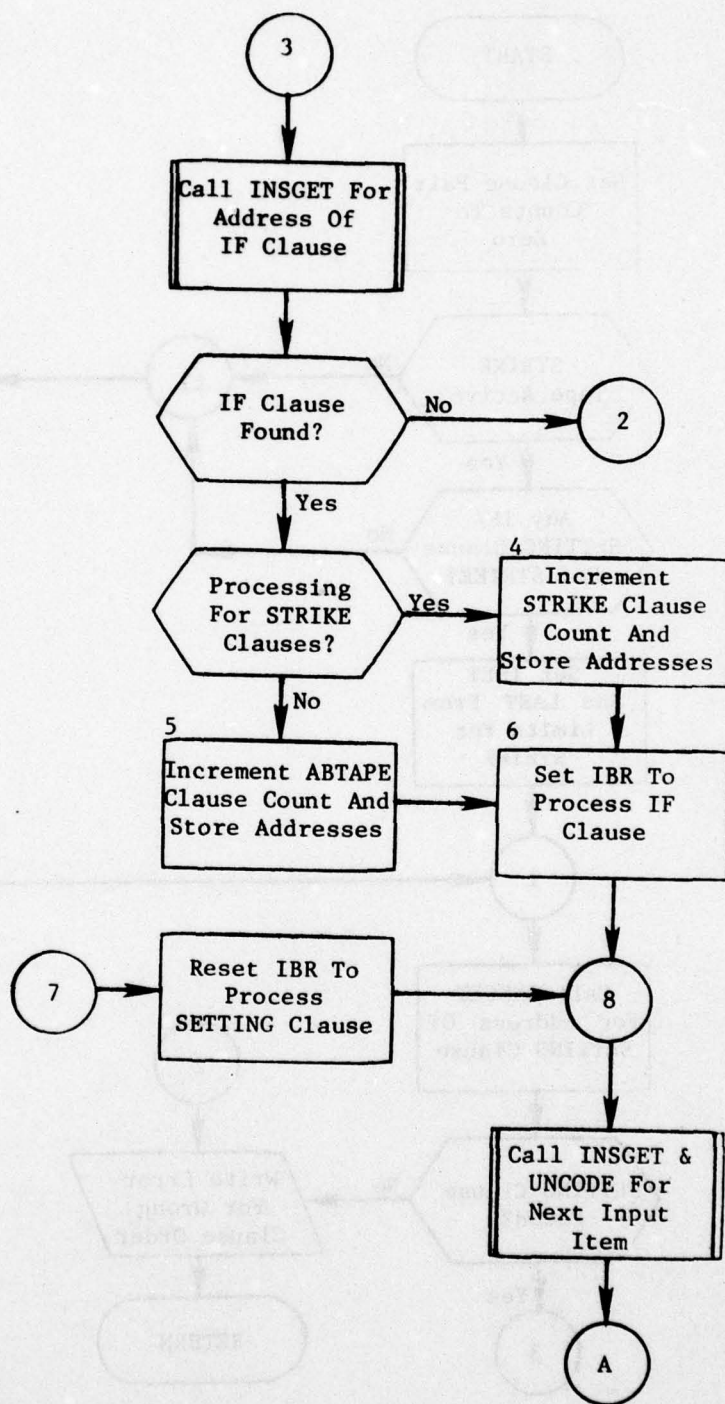


Figure 119. (Part 2 of 4)

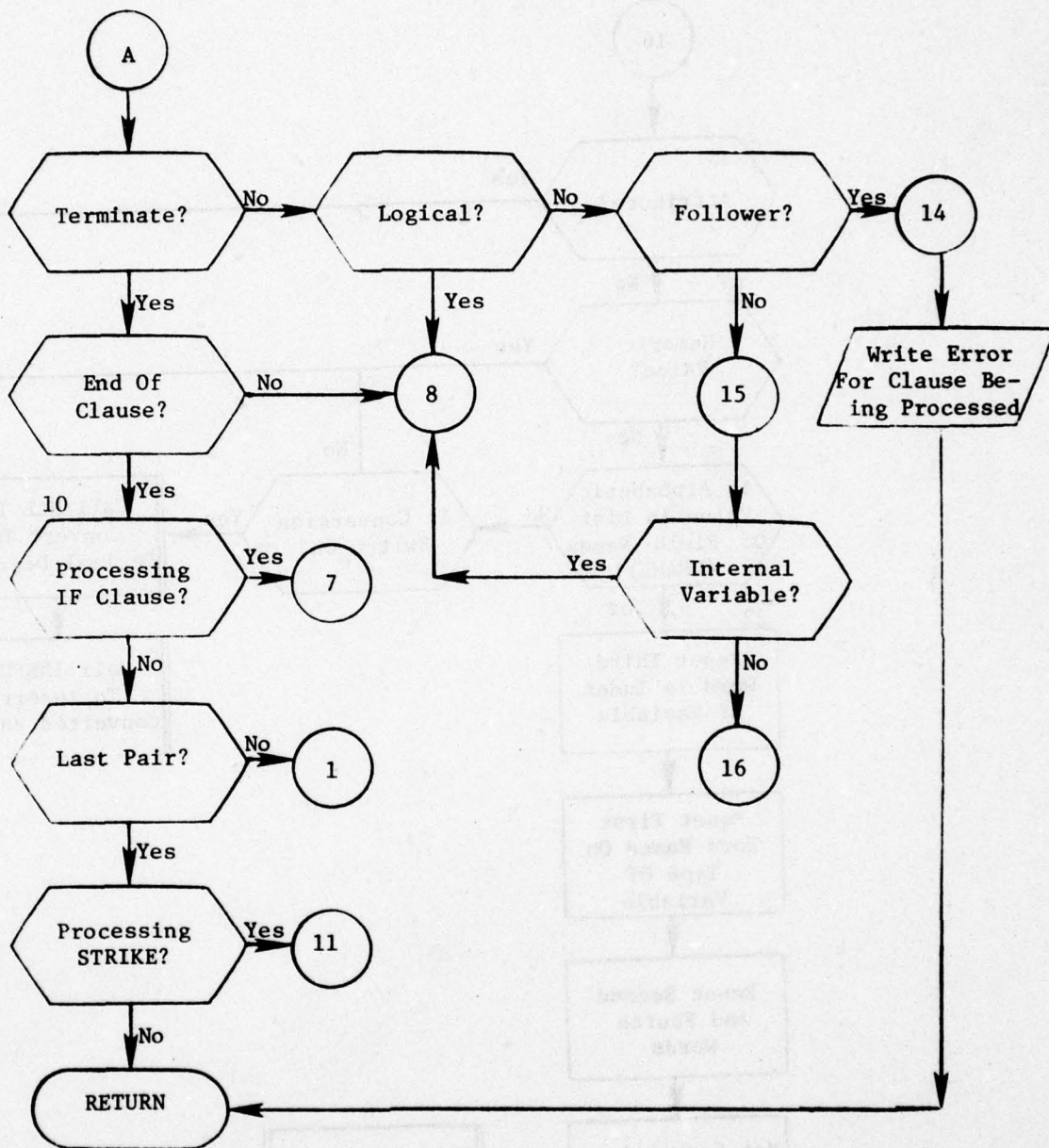


Figure 119. (Part 3 of 4)

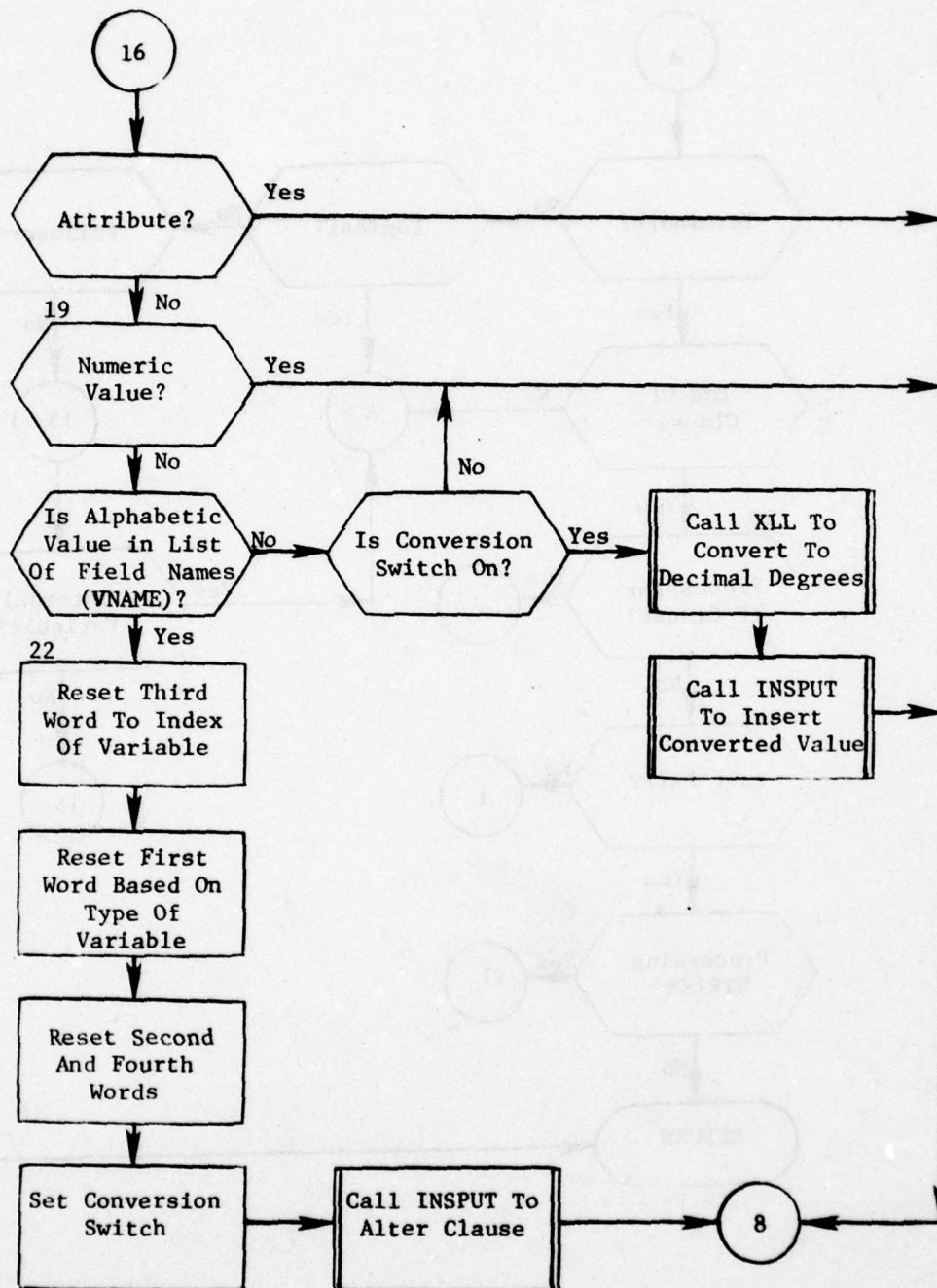


Figure 119. (Part 4 of 4)

4.10.5 Function IFUNCT

PURPOSE: Return numeric function code as per FUNCOM clause.

ENTRY POINTS: IFUNCT

FORMAL PARAMETERS: IXY - Type index of weapon

COMMON BLOCKS: IFUNC, TYPSTF

SUBROUTINES CALLED: None

CALLED BY: ABOUT, STOUT

Method:

Function code value is retrieved from TFUNCT and compared to contents of JFUNC array. When a match is found the corresponding value of INDFUNC is returned.

Function IFUNCT is illustrated in figure 120.

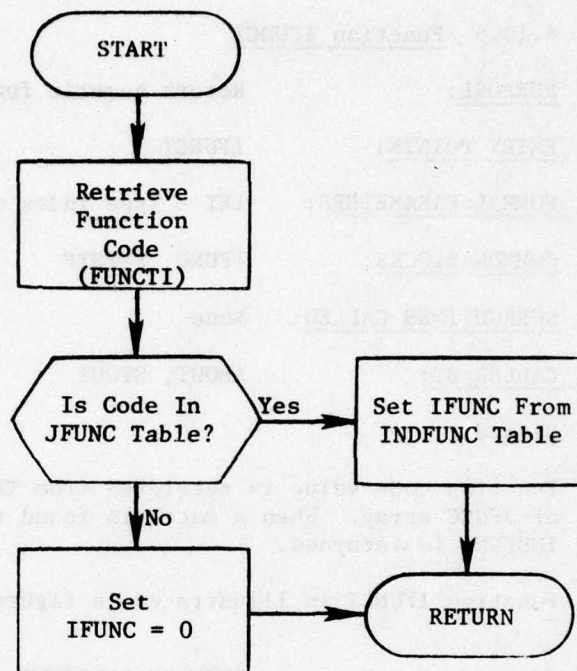


Figure 120. Function IFUNCT

4.10.6 Subroutine INFORM

PURPOSE: To create BCD fields from Numeric input.

ENTRY POINTS: INFORM, IPROB, NTIME

FORMAL PARAMETERS: IN, X, T - Input numeric
OUT - Output BCD field

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: ABOUT, STOUT

Method:

Entry INFORM: Successive digits are obtained from IN by dividing and determining the remainder. These digits are then inserted in OUT from right to left. The effect of a number of fewer than six digits is to zero fill the remainder of OUT.

Entry IPROB: The input is converted to a percent. Percent of 100 or greater are returned as -1. All others are sent to entry INFORM.

Entry NTIME: The input is split into hours, minutes and seconds. It is then packed and sent to entry INFORM.

Subroutine INFORM is illustrated in figure 121.

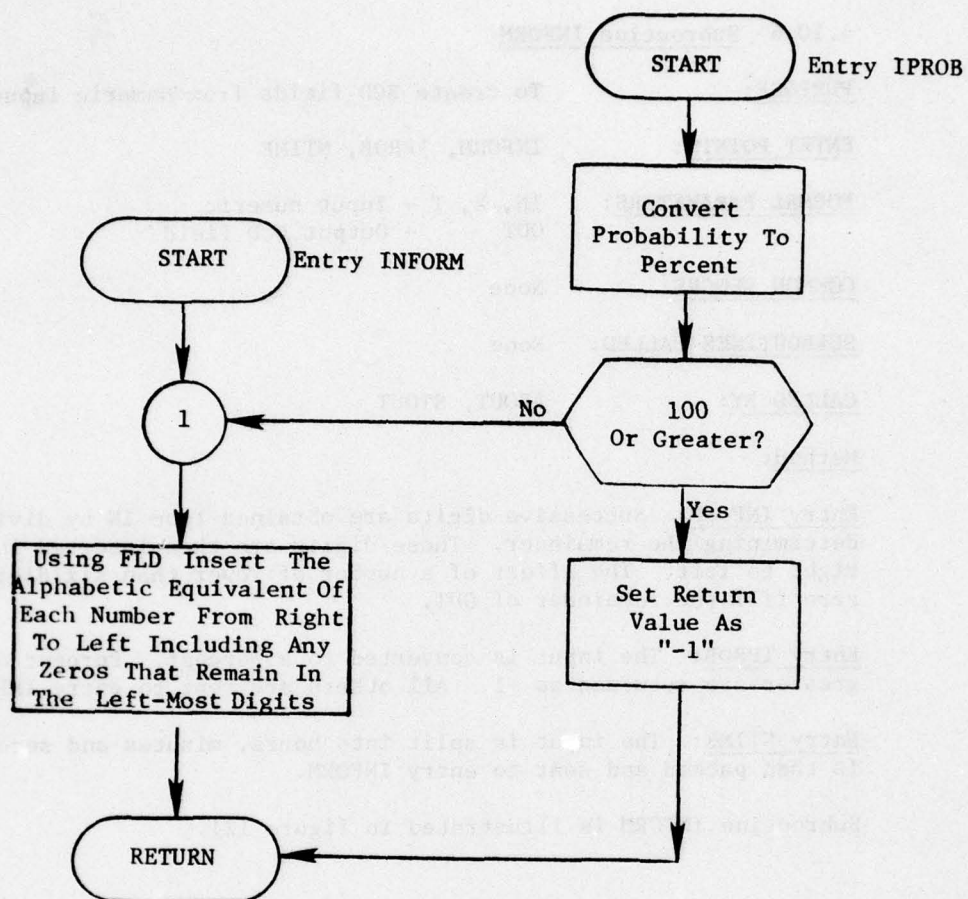


Figure 121. Subroutine INFORM: Entries INFORM and IPROB
(Part 1 of 2)

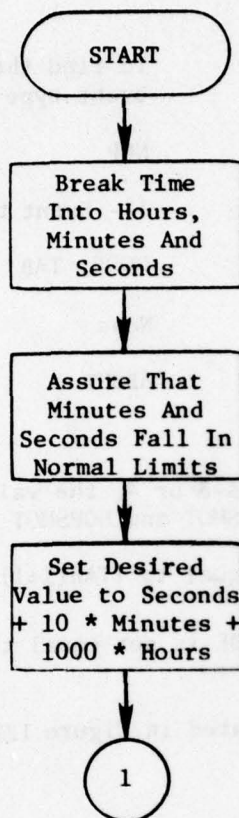


Figure 121. Entry NTIME (Part 2 of 2)

4.10.7 Function NOP

<u>PURPOSE:</u>	To find the operation code associated with event type I.
<u>ENTRY POINTS:</u>	NOP
<u>FORMAL PARAMETERS:</u>	I - Event type
<u>COMMON BLOCKS:</u>	MODE, TAB
<u>SUBROUTINES CALLED:</u>	None
<u>CALLED BY:</u>	ABOUT

Method:

First, if NOPSHOT equals 3 or 4, the values of ITAB(J) for J=18, 19, 20, and 21 are reset to NOPSHOT and NOPSHOT is reset to zero.

In any case NO is set equal to ITAB(I+1),

Finally, if I is 19, MODE is set equal to 4 or if I is 18, MODE is set equal to 1.

Function NOP is illustrated in figure 122.

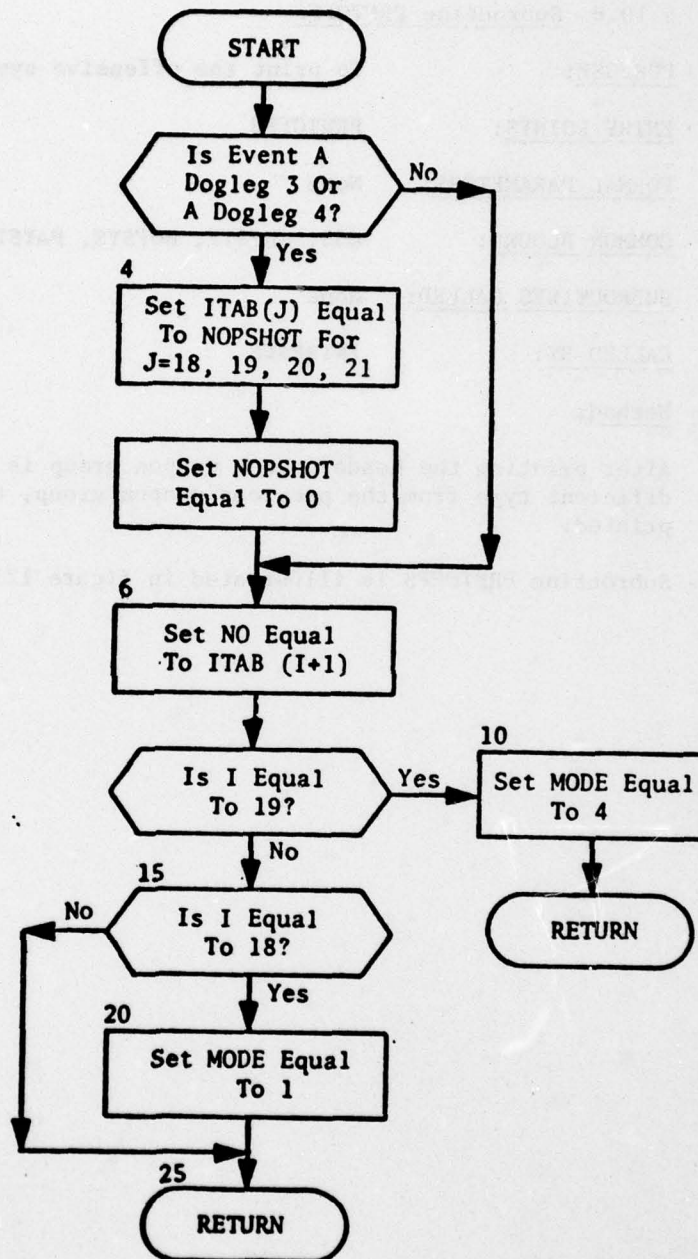


Figure 122. Function NOP

4.10.8 Subroutine PRNTOFFS

PURPOSE: To print the offensive system table.

ENTRY POINTS: PRNTOFFS

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, GRPSTF, NOFSYS, PAYSTF, TYPSTF, WHDSTF

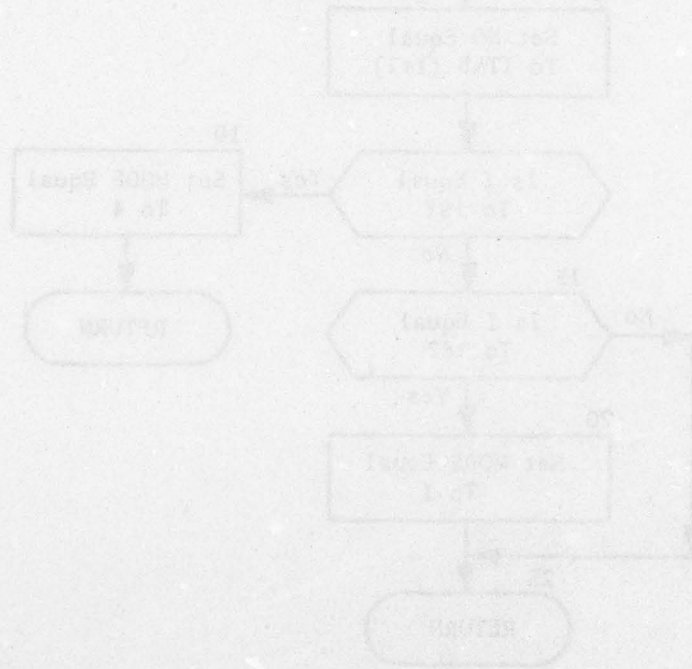
SUBROUTINES CALLED: None

CALLED BY: INTRFACE

Method:

After printing the header, each weapon group is checked. If it is a different type from the previous weapon group, the data for it is printed.

Subroutine PRNTOFFS is illustrated in figure 123.



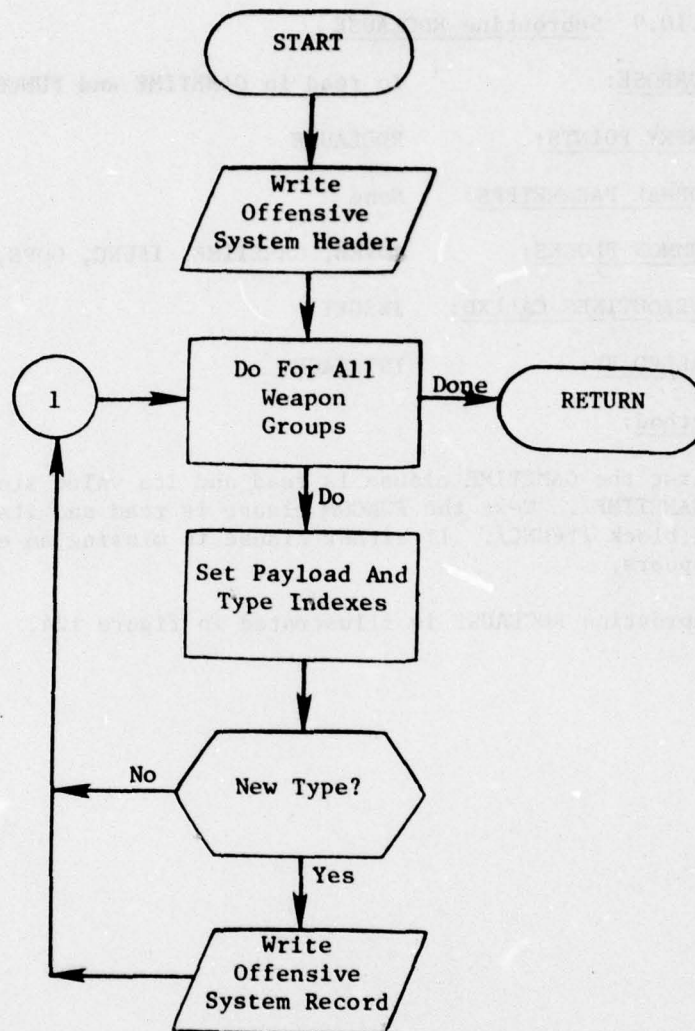


Figure 123. Subroutine PRNTOFFS

4.10.9 Subroutine RDCLAUSE

PURPOSE: To read in GAMETIME and FUNCOM clauses.

ENTRY POINTS: RDCLAUSE

FORMAL PARAMETERS: None

COMMON BLOCKS: ADVRB, GAMETIME, IFUNC, OOPS, ZEES

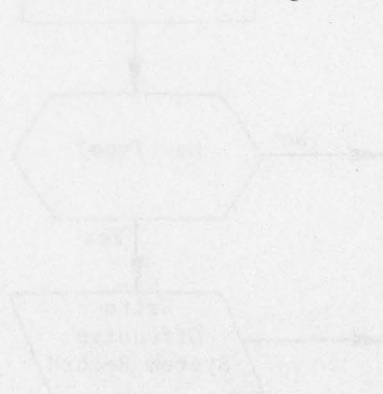
SUBROUTINES CALLED: INSGET

CALLED BY: INTRFACE

Method:

First the GAMETIME clause is read and its value stored in block /GAMETIME/. Next the FUNCOM clause is read and its values stored in block /IFUNC/. If either clause is missing an error message appears.

Subroutine RDCLAUSE is illustrated in figure 124.



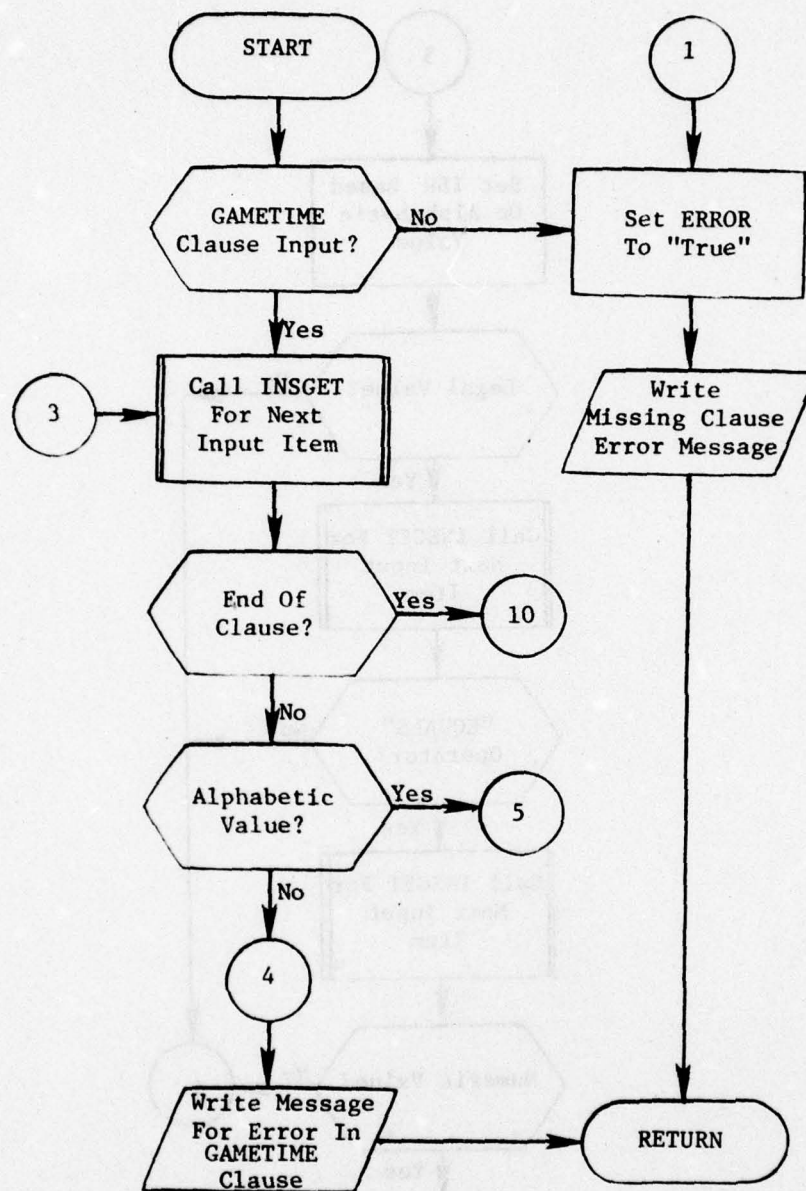


Figure 124. Subroutine RDCLAUSe (Part 1 of 5)

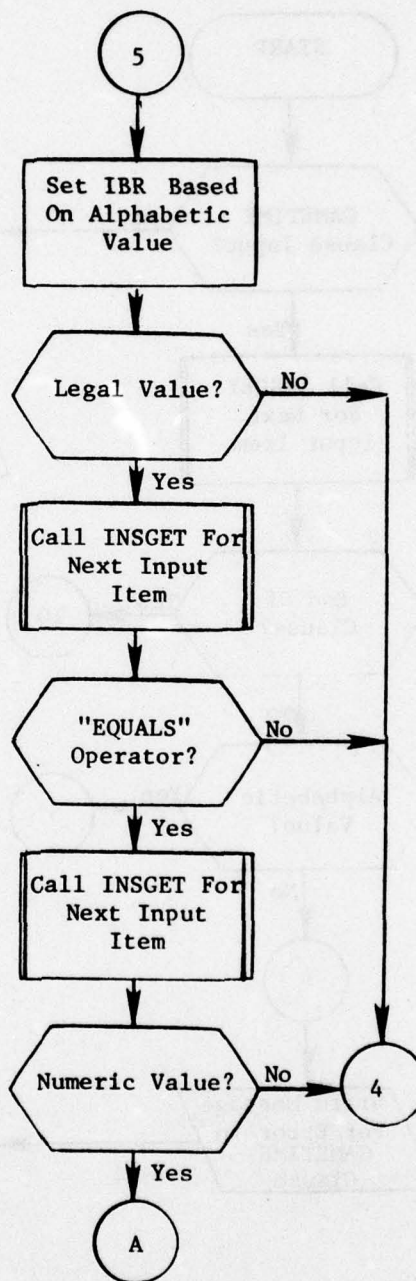


Figure 124. (Part 2 of 5)

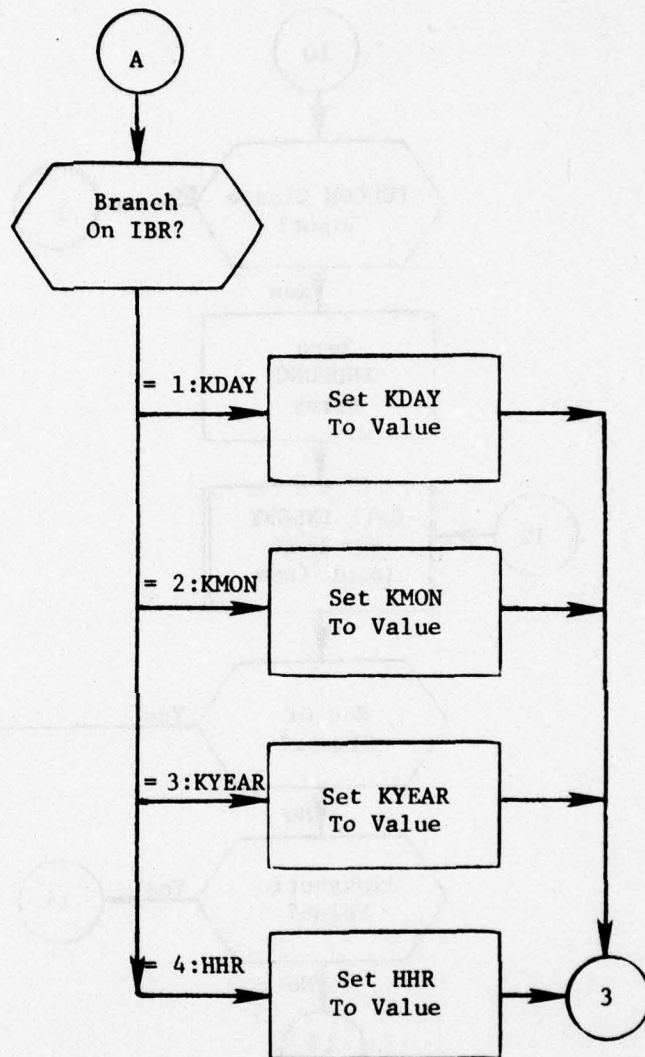


Figure 124. (Part 3 of 5)

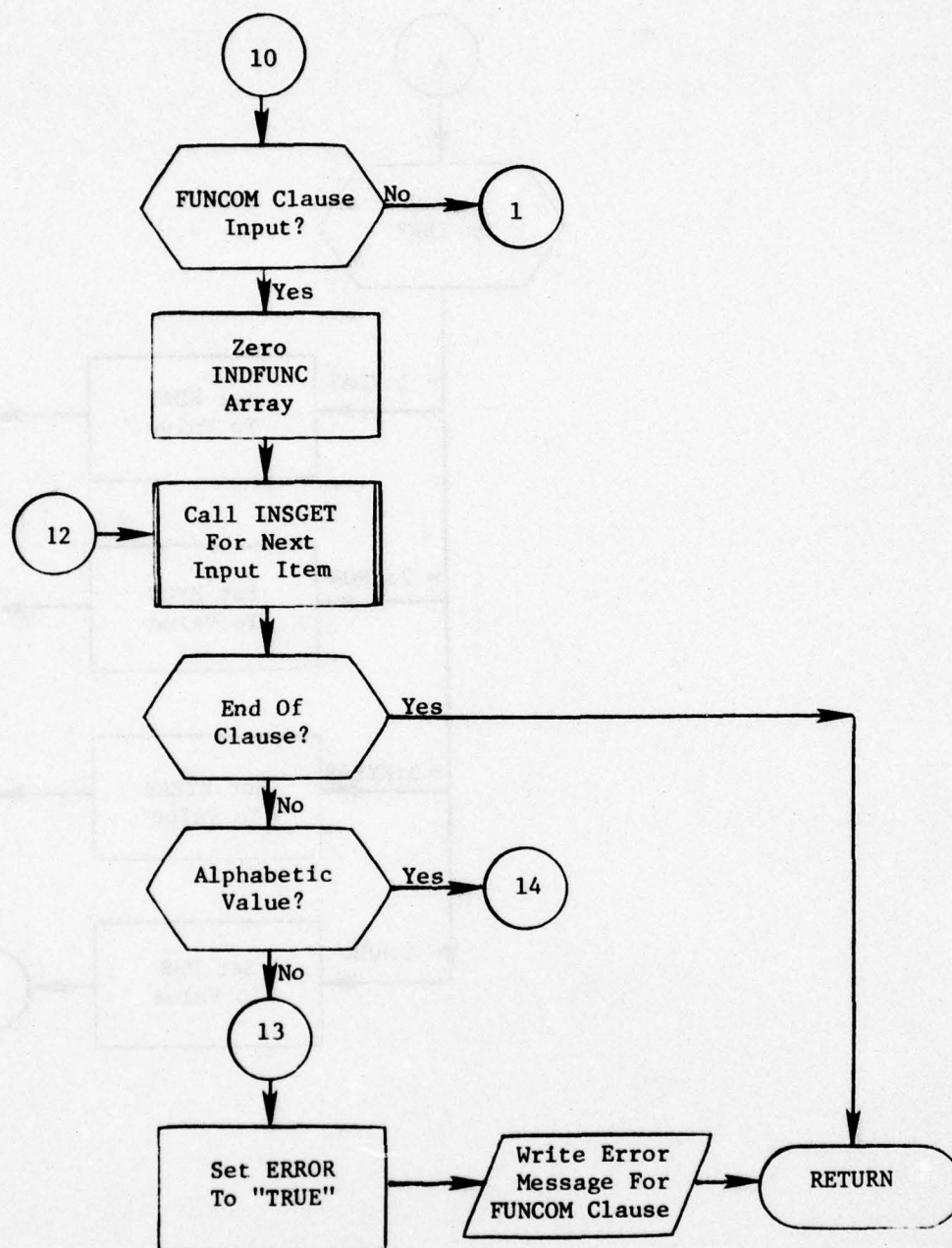


Figure 124. (Part 4 of 5)

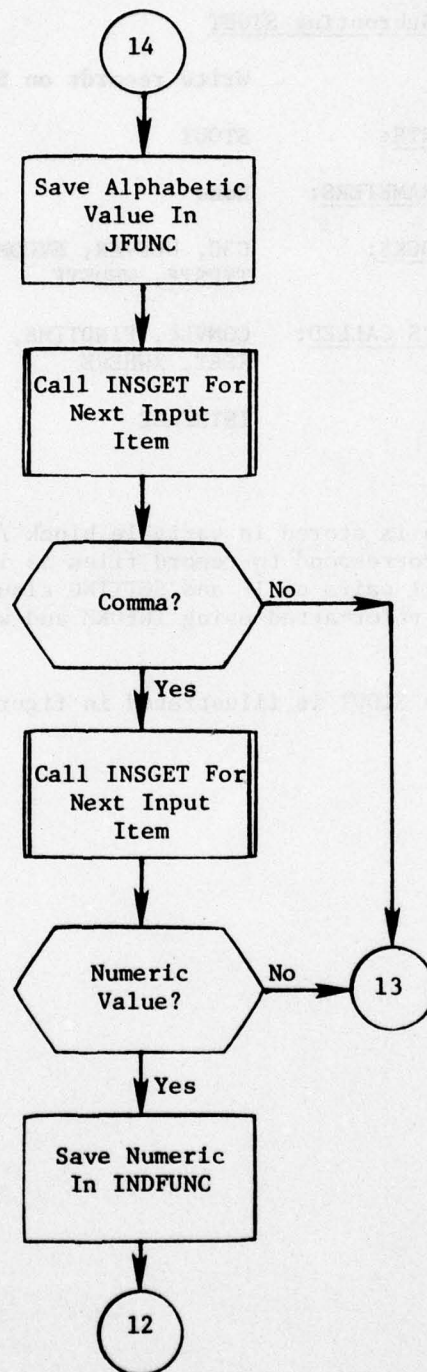


Figure 124. (Part 5 of 5)

4.10.10 Subroutine STOUT

PURPOSE: Write records on STRIKE tape

ENTRY POINTS: STOUT

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, DEFVAR, EVCOM, GRPSTF, IFSCOM, PAYSTF, PSW, TYPSTF, WHDSTF

SUBROUTINES CALLED: CONVLL, FINDTIME, IFUNCT, IGETHOB, INFORM, IPROB, XSET, XWHERE

CALLED BY: INTRFACE

Method:

First data is stored in variable block /DEFVAR/. The positions in /DEFVAR/ correspond to record files as in figure 59. When all data is stored, all pairs of IF and SETTING clauses are executed. Finally, the record is reformatted using INFORM and written on the output unit (LTN4).

Subroutine STOUT is illustrated in figure 125.

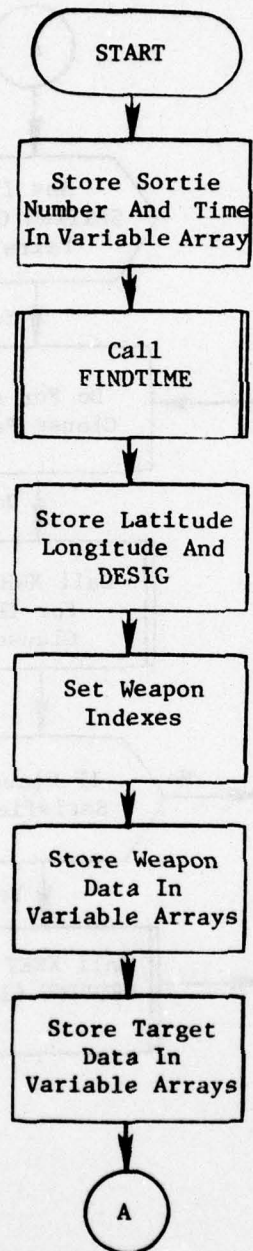


Figure 125. Subroutine STOUT (Part 1 of 3)

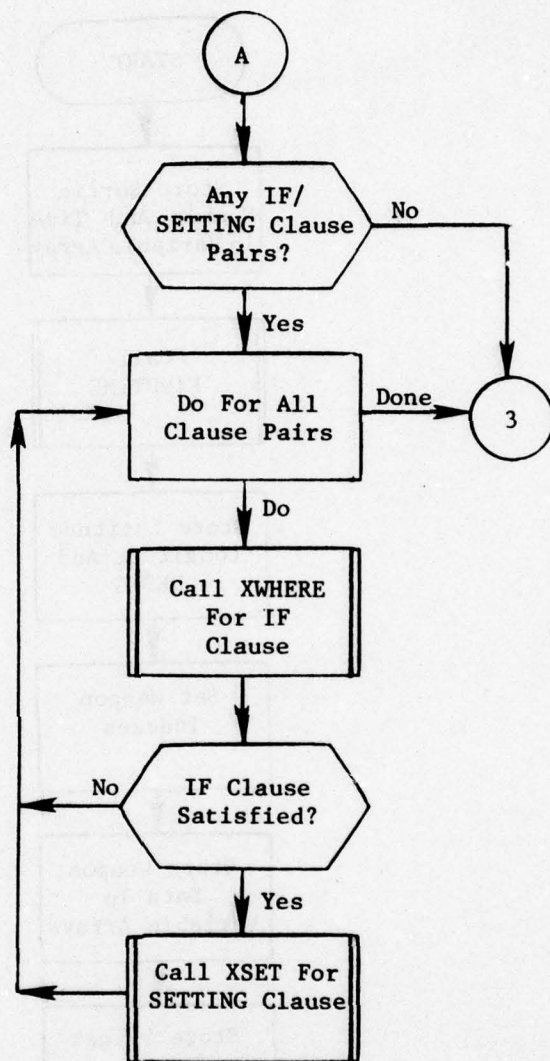


Figure 125. (Part 2 of 3)

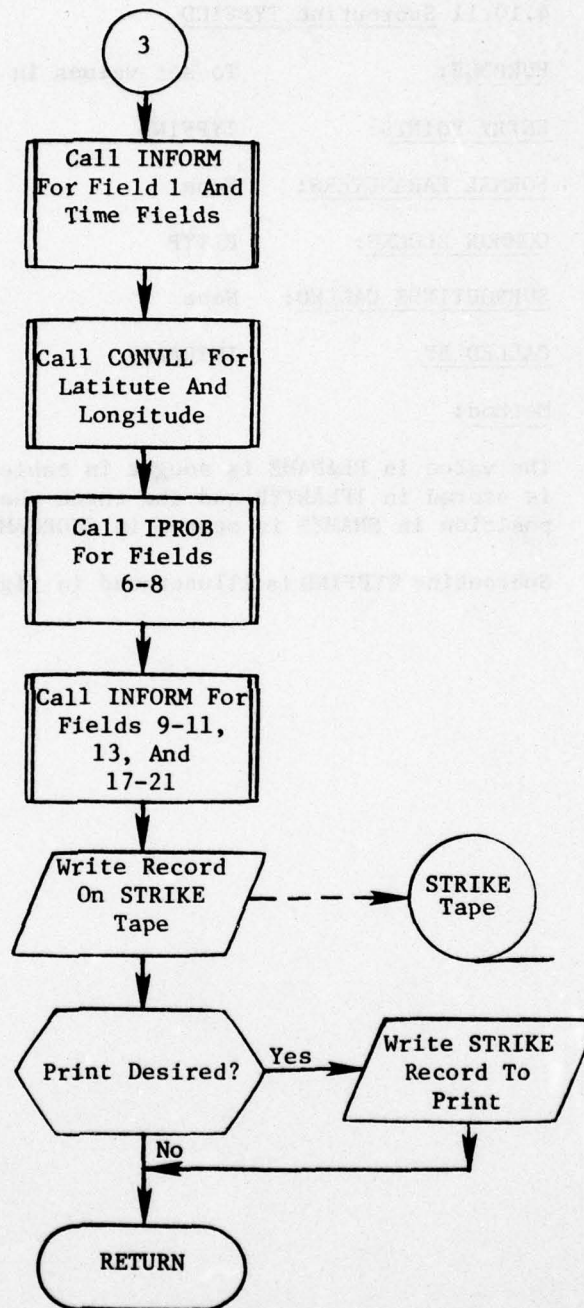


Figure 125. (Part 3 of 3)

4.10.11 Subroutine TYPFIND

PURPOSE: To set values in the /PLTYP/ block.

ENTRY POINTS: TYPFIND

FORMAL PARAMETERS: None

COMMON BLOCKS: PLTYP

SUBROUTINES CALLED: None

CALLED BY: INTRFACE

Method:

The value in PLANAME is sought in table TNames. When found, the index is stored in IPLANTYP and the three character name in the corresponding position in SNames is stored in SHORNAME.

Subroutine TYPFIND is illustrated in figure 126.

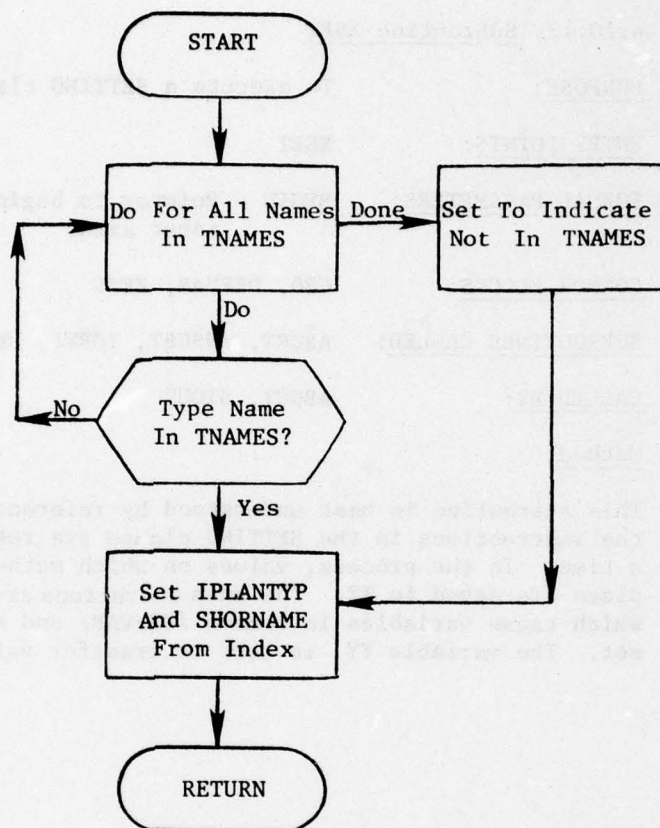


Figure 126. Subroutine TYPFIND

4.10.12 Subroutine XSET

PURPOSE: To execute a SETTING clause.

ENTRY POINTS: XSET

FORMAL PARAMETERS: BEGIN - Pointer to beginning of SETTING clause in input array

COMMON BLOCKS: C30, DEFVAR, ZEES

SUBROUTINES CALLED: ABORT, INSGET, IORFL, UNCODE

CALLED BY: ABOUT, STOUT

Method:

This subroutine is best understood by reference to figure 127. Basically, the instructions in the SETTING clause are retrieved and executed one at a time. In the process, values on which mathematical operations take place are saved in ZZ. The main operations are those of load and equals which cause variables in common /DEFVAR/ and attributes in /C30/ to be set. The variable YY is used to transfer values in this operation.

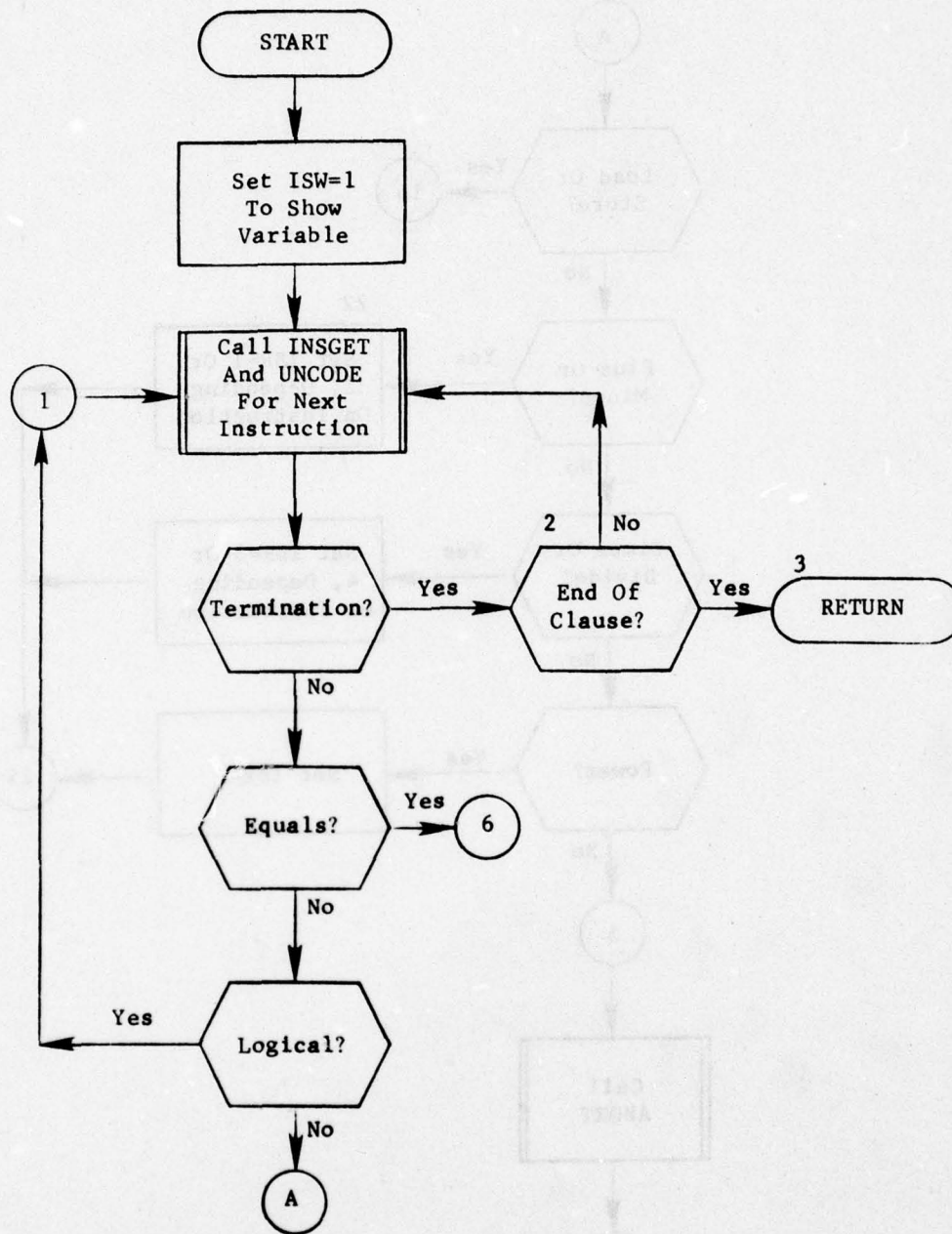


Figure 127. Subroutine XSET (Part 1 of 7)

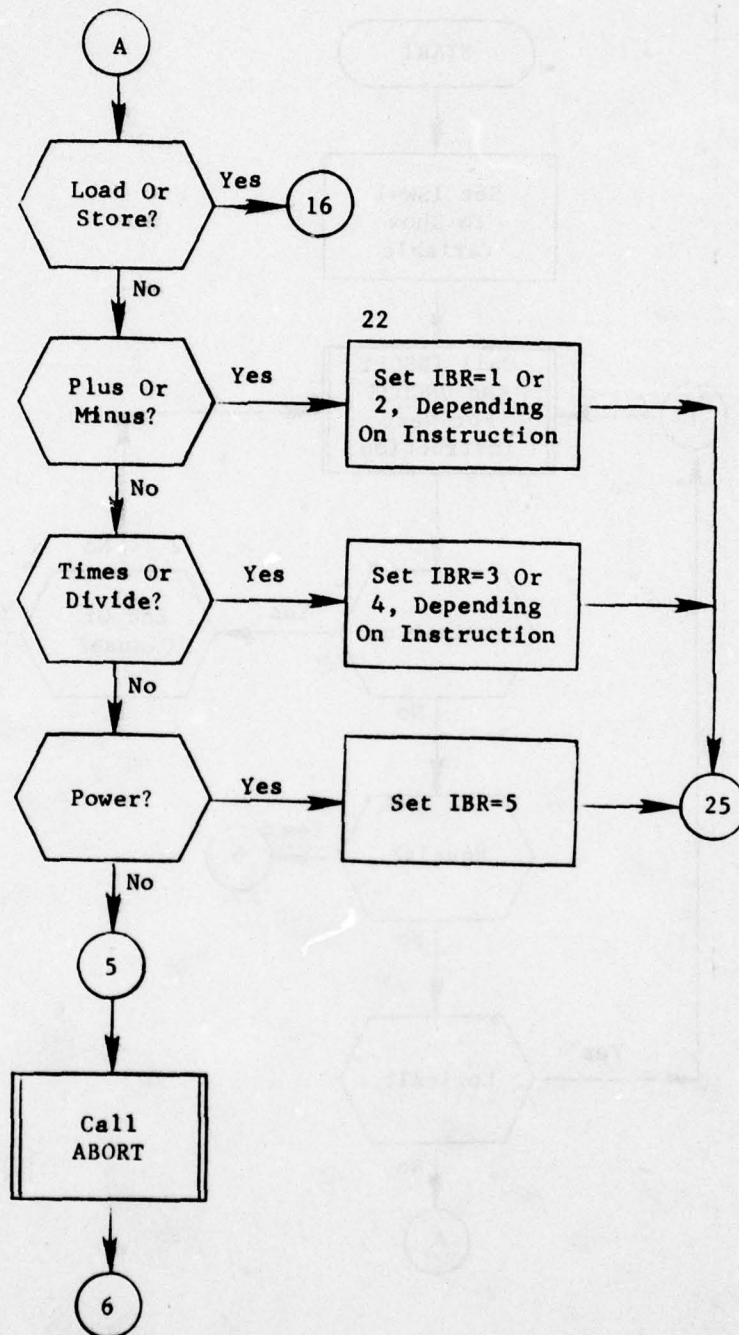


Figure 127. (Part 2 of 7)

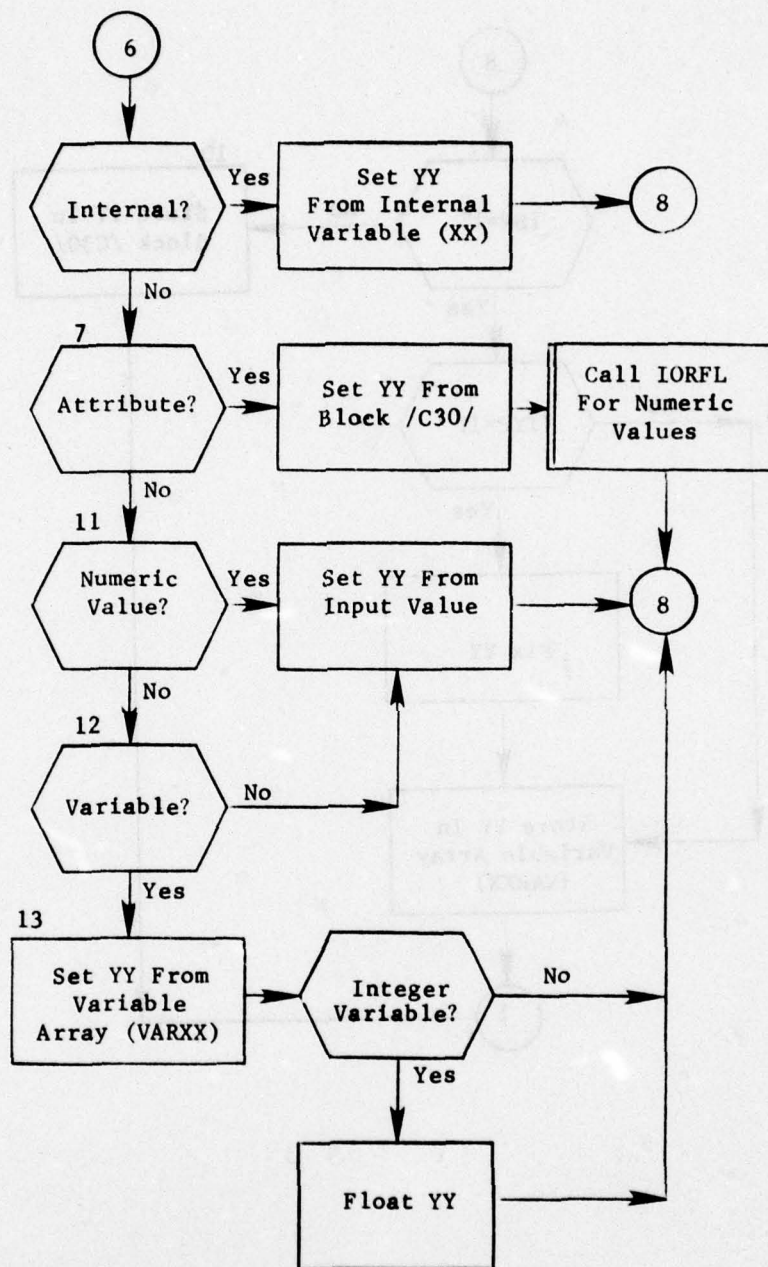


Figure 127. (Part 3 of 7)

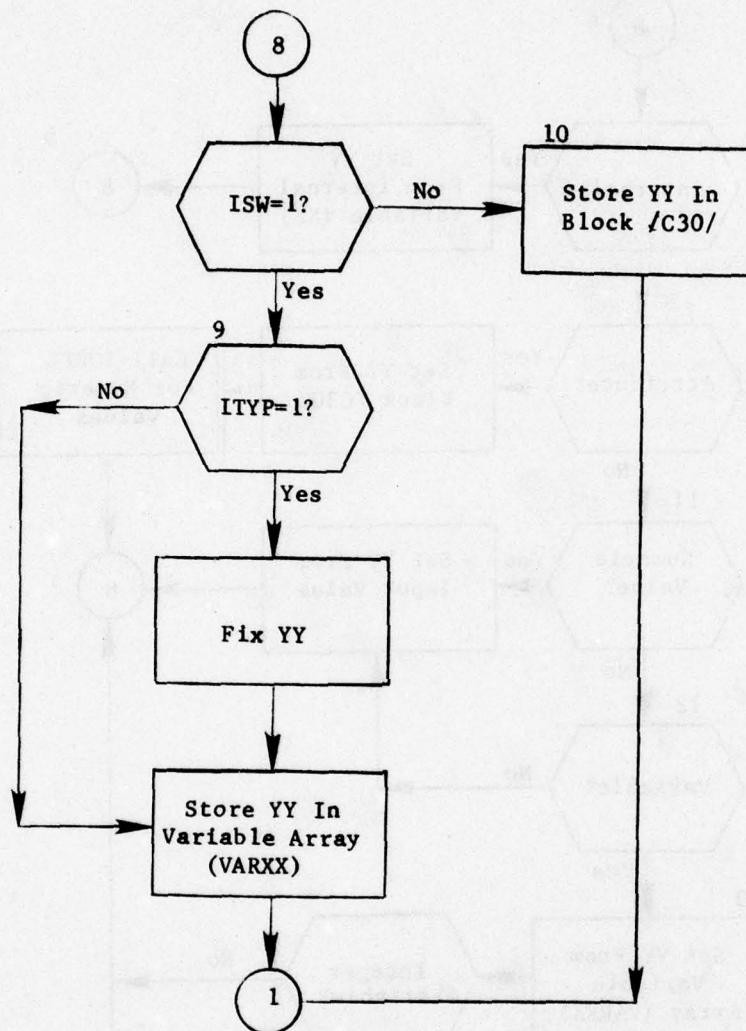


Figure 127. (Part 4 of 7)

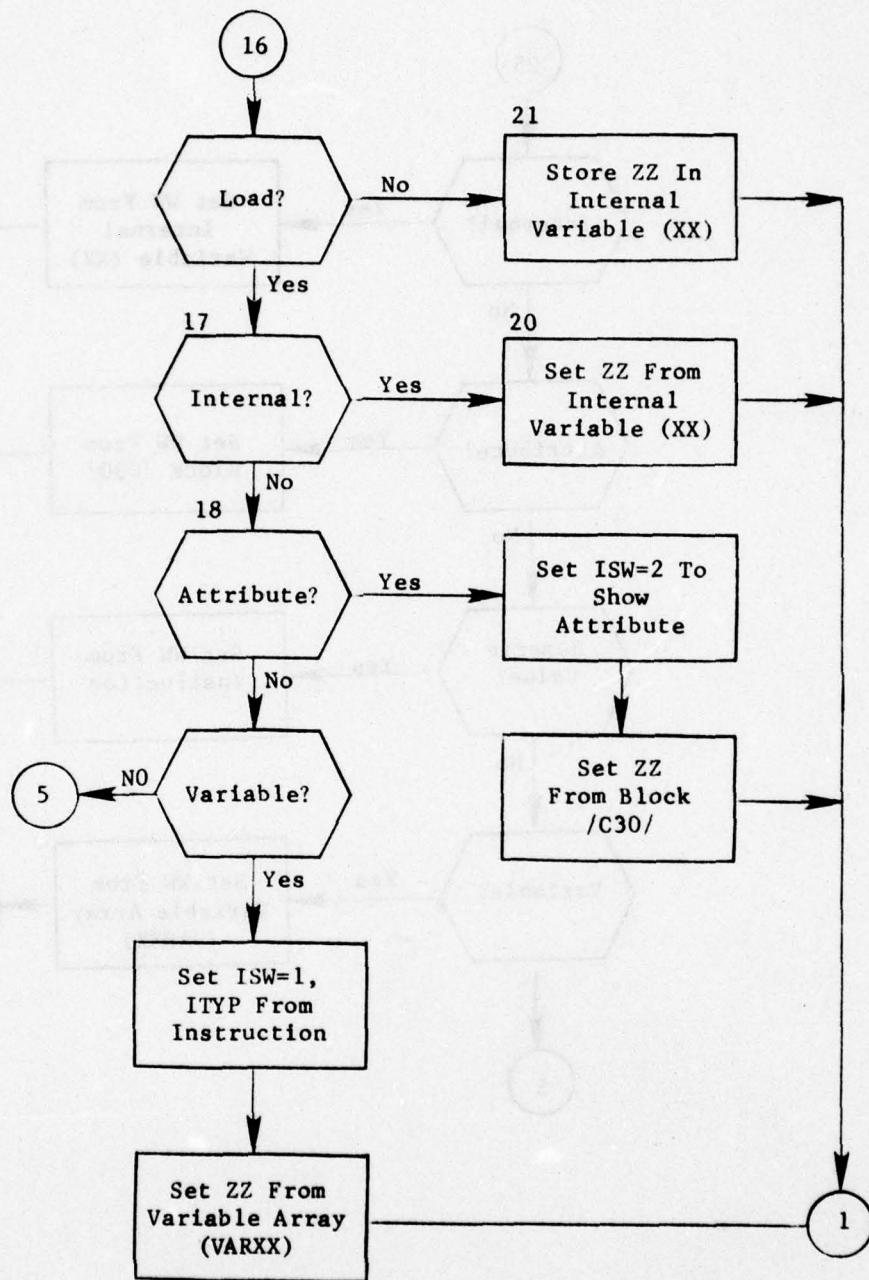


Figure 127. (Part 5 of 7)

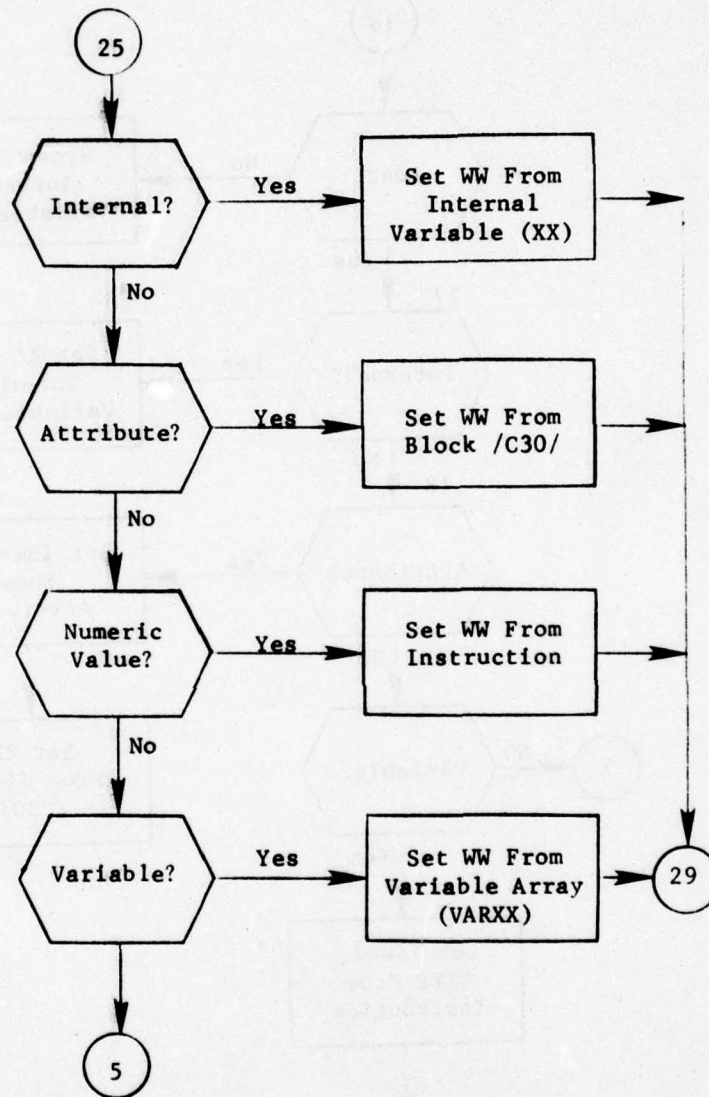


Figure 127. (Part 6 of 7)

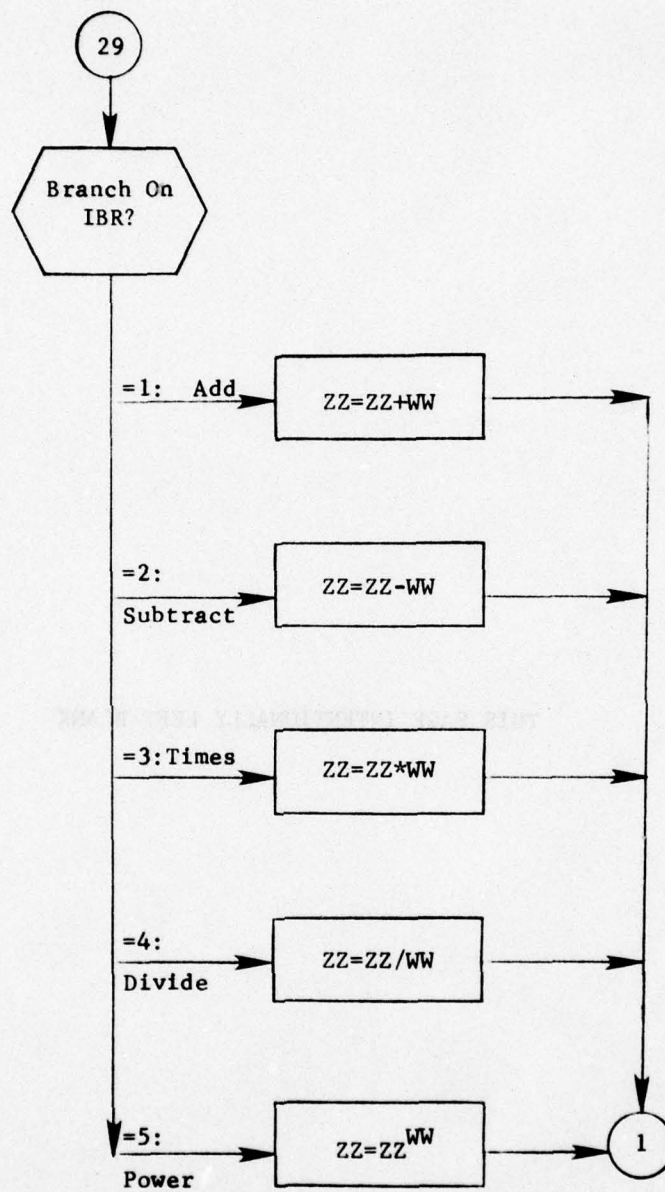


Figure 127. (Part 7 of 7)

REFERENCES

1. M.M. Flood, "The Traveling Salesman Problem," Opns. Res. 4, 61-75 (1956).
2. R.H. Gonzalez, "Solution of the Traveling Salesman Problem by Dynamic Programming on the Hypercube," Interim Technical Report No. 18, OR Center, M.I.T., 1962.
3. M. Held and R.M. Karp, "A Dynamic Programming Approach to Sequencing Problems," J. Soc. Indust. and Appl. Math. 10, 196-210 (1962).
4. M.J. Rossman, R.J. Twery, and F.D. Stone, "A Solution to the Traveling Salesman Problem by Combinatorial Programming," mimeographed.
5. M.J. Rossman and R.J. Twery, "Combinatorial Programming," presented at 6th Annual ORSA meeting, May 1958, mimeographed.
6. W.L. Eastman, "Linear Programming with Pattern Constraints," Ph.D. dissertation, Harvard University, July 1958; also, in augmented form: Report No. BL-20 The Computation Laboratory, Harvard University, July 1958.
7. G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson, "Solution of a Large Scale Traveling Salesman Problem," Opns. Res. 2, 393-410 (1954).
8. G.A. Croes, "A Method for Solving Traveling Salesman Problems," Opns. Res. 6, 791-814 (1958).
9. D.W. Sweeney, "The Exploration of a New Algorithm for Solving the Traveling Salesman Problem," M.S. Thesis, M.I.T., 1963.
10. A. Doig and A.H. Land, "An Automatic Method of Solving Discrete Programming Problems," Econometrica 28, 497-520 (1960).

APPENDIX A

SORTIE GENERATION ALGORITHMS AND CONCEPT

Algorithms employed within the Sortie Generation subsystem are discussed within this appendix. Induced are: basic sortie generation, detailed sortie specifications, actual height of burst calculations, tanker allocation technique and missile timing.

A.1 Basic Sortie Generation

The development of the QUICK strategic war plan may be viewed as incorporating two major planning tasks. The initial task involves that processing required to establish an allocation of weapons to target which maximizes target destruction within the scenario and weapon system constraints established for the plan. Then, to implement this allocation, specific missile and bomber plans (i.e., sortie specifications) must be generated for each delivery vehicle. The latter task, referred to as "sortie generation" includes the preparation of a set of basic sortie specifications and the subsequent expansion/refinement of the data contained therein to produce a set of detailed sortie specifications. This section addresses the development of the basic sortie data. The preparation of detailed specifications is discussed in the following section.

The optimum allocation developed by module ALOC specifies only the weapon type and approximate base location (the group centroid) of the weapons allocation to each target; it does not specify the precise bomber or missile which is allocated to each target. In addition, when allocating bombs and MIRVs (multiple independently targetable re-entry vehicles), ALOC does not consider the requirement for geographically grouping targets for attack by a single delivery vehicle, bomber or MIRV.

The development of the basic sortie data for the individual missiles and bombers (i.e., the generation of the basic sortie* specifications for these vehicles) is primarily performed by module POSTALOC. In the case of missiles, the task is less complex since the missile flight plans are basically determined once a specific target or target set (provided by FOOTPRNT) is associated with a specific type of missile and the launch and target coordinates are known. In the case of bombers, the process is more complicated. The development of basic bomber sorties requires the association of several strikes in a single sortie. Moreover, it is necessary to associate each sortie with specific launch and recovery bases and to select a flight profile which specifies where low-altitude capability should be used. Since the allocator (ALOC) does not distinguish between bombs and air-to-surface missiles (ASMs) carried by the same aircraft, it remains for module POSTALOC to determine which targets should be targeted with bombs and which with air-to-surface missiles.

* As used in QUICK the term sortie refers to an operational flight or flight plan associated with one delivery vehicle, missile or bomber.

Prior to being input to module POSTALOC, the weapon-to-target assignment data developed by module ALOC are processed by module ALOCOUT. The major functions performed by these programs are described in other sections of this manual but are summarized here for purpose of continuity.

The weapon allocator ALOC supplies module ALOCOUT with data for each target, specifying the weapon groups assigned to each target together with associated targeting data. ALOCOUT extracts from these records the data relevant to sortie generation and reorganizes the extracted data by weapon group, giving for each weapon group the number of strikes and the specific targets assigned through each penetration corridor, plus associated data relating to these targets. ALOCOUT is also responsible for computing any aiming offsets required by the plan. In the case of simple targets or multiple targets, these offsets are simply set to zero. In the case of complex targets, which can have several elements at slightly different coordinates, ALOCOUT selects optimum aim points within the target complex. A complex target (or target complex) is a combination of target elements sufficiently close that a weapon on any one of them will have some probability of killing other elements in the complex. Such target complexes must be targeted as a unit -- not as individuals. Thus, module ALOC treats them as a unit, allocating weapons against their total value, using one set of coordinates. In order to maximize targeting efficiency against such a complex, one must select desired ground zeros (DGZs) or aim points among the target elements.

If the plan includes missile weapon groups equipped with MIRVs (multiple independently targetable re-entry vehicles), module FOOTPRNT must be included in the plan development cycle. This program processes the individual weapon-to-target assignments and constructs the specific booster loads (the re-entry vehicle-to-target assignments to be associated with a single MIRV-capable missile) for each weapon group with a MIRV capability.

Bomber Plans

The sortie definitions developed in module POSTALOC are generated separately for each weapon group and, within each weapon group, separately for each penetration corridor. For tactical bombers or naval bombers (i.e., PKNV > 0.0), a penetration corridor is not used. However, to preserve the logic of the program, a dummy corridor index is defined to indicate no corridor usage. This corridor index is tested before performing distance calculations and strike assignments so that the appropriate substitutions are made in the method of processing. The basic sortie plan consists of ordered lists of the targets to be struck by each bomber, an indication of whether a target is to be struck with a bomb or an ASM, and an estimate of the distances between successive flight points that are flown at low altitude. The sortie definition does not, however, include the actual coordinates for the various events; e.g., launch, refuel, and drop bomb. These, together with the release points for ASMs are calculated in module PLANOUT.

The bomber sorties are actually constructed in the following fashion. First, the module reads in the strikes assigned to a given group. However, it reads them one corridor at a time. This division of strikes forms a raid; i.e., the aircraft from one group routed by way of not more than one corridor. Next, the strikes in the raid are roughly divided among the available vehicles and bases. Then, each sortie is evaluated in considerable detail, taking into account bomber range, estimated attrition rates, low altitude capability, and the indicator used in weapon allocation which specifies the use of either bombs or ASMs on a given target. During this process, provision is made to omit strikes that seem unprofitable. Each strike omitted may be assigned to another sortie, so that this phase usually includes some refinement of the initial rough allocation of strikes. Only after all of the sorties for the given corridor are defined are the strike data for the next corridor read in. Rather than omit some strikes, the module will consider using an ASM on a bomb target (or a bomb on an ASM target) if there are ASMs (or bombs) available. Although this use of an ASM or bomb is contrary to the allocation preference, this use of weapons is preferable to omitting strikes with no replacement.

A more detailed discussion of initial raid generation and sortie optimization is included below.

Initial Raid Generation: As indicated above, the first step in the generation of the sorties for a given weapon group and corridor is to ascertain the portion of the vehicles and warheads in the group that should be allocated to each raid. For a first approximation, the number of warheads assigned to each penetration corridor is proportional to the number of strikes assigned in each corridor in module ALOC. However, if this number of warheads does not correspond to an integral number of delivery vehicles, the necessary additional warheads required to produce an integral number of delivery vehicles are assigned to each corridor as it is processed. Since the corridors are delivered for processing in order of decreasing number of strikes assigned, this rule puts a slightly higher ratio of bombers to targets in corridors with large raids. In this way, bombers assigned to corridors where there are few other bombers will have more flexibility to select from the geographically sparse target set assigned. In the extreme case where a corridor happens to have only one or two isolated strikes assigned, the corridor will probably be skipped in the assignment of bombers from the group, so that isolated individual bombers are less likely to be assigned to such a corridor.

The next necessary task is to assign strikes within the raid to individual sorties. This requires the assignment of individual weapons to individual targets in accordance with the location of the targets relative to the penetration corridor. The assignment is accomplished through the use of curvilinear coordinate systems chosen to parallel typical flight paths within the penetration corridor.

Figure 128 illustrates two examples of the coordinate system employed in the planning of corridor penetrations. For strategic bombers, the coordinate system shown is established with the $X=0, y=0$ position corresponding to the origin of the penetration corridor. The y axis is parallel to the axis defined by the corridor origin and the coordinates of the corridor orientation point. For tactical or navel bombers, the $x=0, y=0$ position is defined as follows. Consider the centroid of the group of launch bases and the centroid of the group of target bases; and define the distance between the two centroids to be $DISTC$. The origin of the coordinate system is located at the end of the directed line segment which originates at the target centroid, passes through the launch base centroid, and has a magnitude of $2 \times DISTC$. Thus, in this coordinate system it is possible to locate both the targets and the launch bases.

The equations which describe the transformation from the Cartesian coordinates x, y to the curvilinear coordinates ρ, ϕ , are as follows:

$$\phi = \frac{x}{y^k}$$

$$\rho = y^2 + kx^2 \quad \text{for } |\phi| \leq 1.0$$

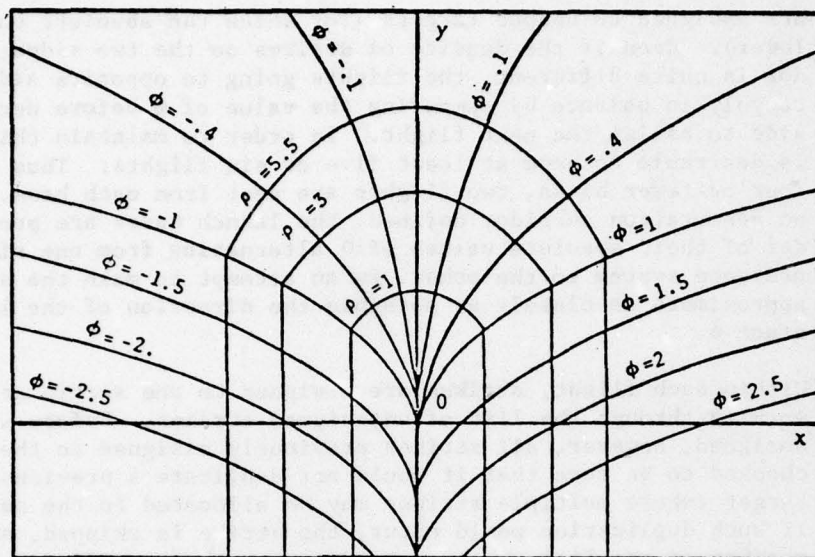
$$\phi = \begin{cases} 1 + |x|^{1/k} - y & x \leq 0 \\ -(1 + |x|^{1/k} - y) & x > 0 \end{cases}$$

$$\rho = |x|^{2/k} + kx^2 \quad \text{all } x$$

Investigation of the two graphs presented reveals that lines corresponding to constant values of ϕ roughly parallel the type of flight paths which should be followed by penetrating bombers. Thus, in the assignment of sorties, a single bomber should be assigned targets which have approximately the same values of ϕ . Further consideration of the graphs indicates the alteration of the parameter k can be used to reflect certain planning objectives into the sorties. For example, higher values of k should be used when saturation of defenses is desired, while lower values should be used if greater importance is attached to minimizing the flight distances to targets (k is the corridor parameter $KORSTY$).

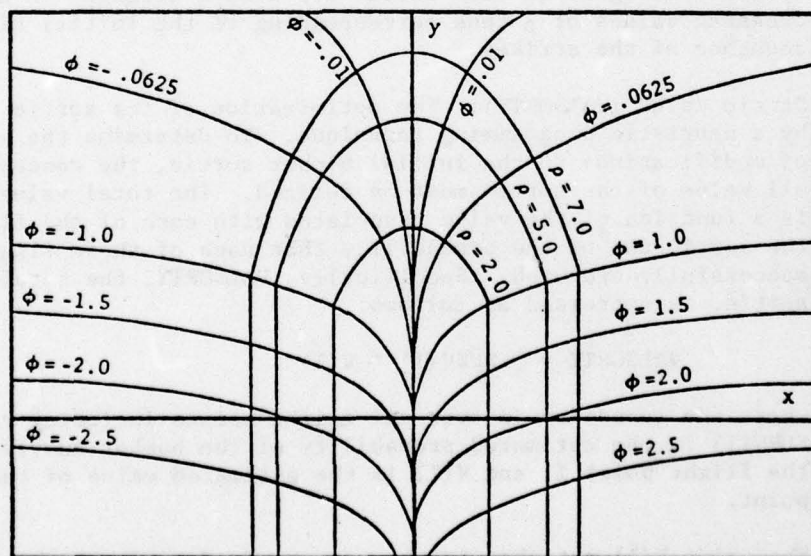
The procedure used to assign strikes within the raid to individual sorties now becomes clear. First, all strikes are arranged in increasing order of their ϕ coordinates. Then, the flights from each launch base are processed in order of the distance from the base to the corridor entry point, thus causing the vehicles to be processed in order of time of arrival.

To provide an approximation of saturation and roll-back tactics, each flight is assigned, as a unit, to either one side of the corridor or the other. The first flights are usually assigned to shallow targets (for which the absolute values of ϕ are higher), while later flights



(A)

K=2



(B)

K=4

Figure 128. Illustrative Curvilinear Functions

are assigned to deeper targets (for which the absolute values of ϕ are lower). Even if the density of strikes on the two sides of the corridor is quite different, the flights going to opposite sides are kept roughly in balance by comparing the value of ϕ before deciding to which side to assign the next flight. In order to maintain this balance, it is desirable to have at least five or six flights. Thus if there are four or fewer bases, two flights are sent from each base. If there is no penetration corridor defined, the launch bases are processed in order of their absolute values of ϕ alternating from one side of the coordinate system to the other, in an attempt to make the sortie paths approximate as closely as possible the direction of the lines of constant ϕ .

Within each flight, strikes are assigned to one sortie at a time by working through the list of unassigned strikes. Before any strike is assigned, however, all strikes previously assigned to the sortie are checked to be sure that it would not duplicate a previously assigned target (where multiple strikes may be allocated to the same target). If such duplication would occur, the strike is skipped, and later strikes on the list are processed to get the specified quota for the sortie. Processing for the next sortie in the flight always begins with the first unassigned strike and continues from there. Strikes actually assigned to each sortie are always arranged in order of increasing values of ρ thus corresponding to the initial time order or sequence of the strikes.

Sortie Value (VALSORTY): The optimization of the sortie is accomplished by a heuristic programming technique. To determine the effectiveness of modifications to the initial bomber sortie, the concept of the overall value of the sortie must be defined. The total value of the sorties is a function of the value associated with each of the flight points in the sortie and of the probability that each of these flight points is successfully executed. Specifically, VALSORTY, the total value of the sortie, is expressed as follows:

$$\text{VALSORTY} = \sum \text{SURV}(I) * V(I)$$

where the summation is over all flight points including recovery. $\text{SURV}(I)$ is the estimated probability of the bomber surviving to reach the flight point I , and $V(I)$ is the estimated value of reaching that point.

The value $V(I)$ attached to the target, I , depends on whether it is to be attacked by a bomb or an ASM.

1. If I is target for a bomb then: $V(I) = \text{RVAL}(\text{tgt})$
2. If I is target for an ASM then: $V(I) = \text{RVAL}(\text{tgt}) * [1.0 + \text{TIMEPREM}(\text{tgt})]$

In the second relation, TIMEPREM is a bonus factor that is

given for using an ASM on targets selected for ASMs in the weapon allocation process.

3. If I is a recovery point then we define: $V(I) = \text{VALRECVR} * \sum \text{RVAL}(\text{tgt})$. In the third equation, the summation is over all targets in the sortie, which implies that the value of recovery is equal to the input parameter fraction VALRECVR of the value of all targets in the mission.

The variable RVAL as calculated in program ALOC is actually a measure of the marginal utility of each weapon. For weapon allocations not directed by the player (not allocated through the use of the "fixed assignment" capability), the marginal utility RVAL is computed as

$$\text{RVAL}_J = \left[\frac{(\text{VTD}_{I-1} - \text{VTD}_I)}{\lambda_I} \right] / \text{PEN}_J$$

where:

VTD_I = Residual target value after the allocation of the I^{th} weapon(s) on target J
 $[\text{VTD}_0 = \text{VTO} = \text{original target value}]$

λ_I = Lagrange multiplier for the I^{th} weapon

PEN_I = Aggregate penetration probability for I^{th} weapon

This formula applies for targets with no terminal missile defenses. In this instance, VTD_{I-1} is equal to the residual target value prior to the allocation of the I^{th} weapon. However, for targets with terminal ballistic missile defenses, VTD_{I-1} is defined to be the residual target value if all weapons from the same group as weapon I are removed. This affords an accurate representation of missiles which are used for defense suppression.

For all weapons assigned by the fixed weapon assignment capability, the marginal utility is computed as

$$\text{RVAL} = \text{VTO} / \lambda_i$$

The computation of $\text{SURV}(I)$ for the formula is based on a simple exponential attrition law. If the integrated attrition probability on each individual leg to a point J is given by $\text{ATLEG}(J)$, then the survival probability for the bomber to the point I will be given by:

$$\text{SURV}(I) = \text{EXP} \left[- \sum_{J=1}^{\text{J} \neq I} \text{ATLEG}(J) \right]$$

The attrition ATLEG(J) includes both area and local attrition for the leg.

Application of Low-Altitude Range: In selecting low-altitude range, QUICK assumes that on any leg or fraction of a leg flown at low altitude the attrition rates will be reduced by the factor HILOAT. In order to estimate the expected value of the sortie, therefore, an estimate must be made of how the available low-altitude range should be applied. Notice that a change in the assumed attrition rate for any leg or part of a leg will change the integrated attrition for the leg ATLEG(J). This in turn will change the probability of survival to any point I (SURV(I)) which is required to evaluate VALSORTY.

The program therefore begins by summing the total distance for the sortie as specified. This distance is subtracted from the aircraft range to give the surplus range RNGSURP available for the mission. Using the conversion factor RANGED, this surplus range is used to estimate the available low-altitude distance AVAILOW for the mission. Finally, AVAILOW is allocated to the various legs.

The allocation of the low altitude range AVAILOW is subject to several restrictions. In order to realistically model actual bomber flight profiles, low altitude range is allocated so that the bomber can go from low to high altitude only once after it passes the corridor origin. Attribute PAYALT (bomber weapon release altitude indicator) can also restrict the low altitude range allocation. If the value of attribute PAYALT is HIVAL (its default value), then low altitude range is allocated in a manner intended to maximize the value of the sortie VALSORTY. Weapon releases (i.e., bomb drops or ASM launches) may occur at either high or low altitude. However, if PAYALT was the value LOW, all weapon releases must occur at low altitude. In this case the available range is first allocated to the intertarget legs. If the bomber does not have sufficient fuel reserve to fly over all weapon release points at low altitude, weapons are deleted until this criterion can be met. If there is more low altitude range available after the intertarget allocation, the excess is allocated to maximize sortie value. The allocation of this excess is performed in the same manner as in the HIVAL case. If PAYALT has the value HIGH, all weapon releases must occur at high altitude. The available low altitude range, if any, is first allocated to bomber flight prior to the first target (as in the HIVAL case) to maximize sortie value. A low to high altitude transition is then planned just prior to the first target so that all weapons can be released at high altitude. If the bomber has sufficient fuel reserves that it can fly the entire distance from the last target to recovery at low altitude, then a high to low altitude transition is planned following the last target. There is no corresponding low to high altitude transition planned for this second low altitude flight because the bomber is allowed to climb to high altitude only once after penetration. In this case, the bomber proceeds to the recovery base at low altitude. For the remainder of this section, the low altitude range allocation method assumes that PAYALT has value HIVAL. For other values of PAYALT, the altitude

constraints are imposed over the target area and any excess range is allocated as described below.

During this allocation of available low-altitude range, the following alternatives are provided:

- a. Allocate low-altitude range to that remaining precorridor leg that has the highest attrition
- b. Extend the low-altitude flight from the first target one more leg toward the depenetration point (where the attrition is assumed to end)
- c. Extend the low-altitude flight a little further in front of the first target toward the corridor origin.

Choices among these alternatives are made on the basis of which one will produce the largest rate of increase in VALSORTY per nautical mile of low-altitude range required.

To illustrate how the priorities for this allocation work out mathematically, we note that the cumulative survival probability SURV to route point i can be represented as a product of the survival probabilities S_j for each leg j up to and including the i th. Thus we can rewrite the equation for VALSORTY as follows:

$$V = \sum_{i=1}^{i=n} \left\{ \prod_{j=1}^{j=i} S_j \right\} V_i$$

where V is the value of the sortie and V_i is the value of successfully reaching the i th route point. (This is referred to as the value done, or VALDONE, in the program.)

We also note that $S_j = e^{-\alpha_j}$ where α_j is the total attrition of the j th leg. Obviously α_j is a function of L_j , the low-altitude distance allocated to the j th leg.

Differentiating V with respect to L_k , the low altitude allocated to some specific leg k , we obtain

$$\frac{\partial V}{\partial L_k} = \frac{\partial V}{\partial S_k} \frac{\partial S_k}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial L_k}$$

while

$$\frac{\partial V}{\partial S_k} = \sum_{i=k}^{i=n} \frac{1}{S_k} \left\{ \prod_{j=1}^{j=i} S_j \right\} V_i$$

$$\frac{\partial S_k}{\partial \alpha_k} = -e^{-\alpha_k} = -S_k$$

Thus

$$\frac{\partial V}{\partial L_k} = - \left[\sum_{i=k}^{i=n} \left\{ \begin{matrix} j=i \\ \pi \\ j=1 \end{matrix} S_j \right\} v_i \right] \frac{\partial \alpha_k}{\partial L_k}$$

Now separating out the common factors S_j for $j=i,k$, and noting that

$$\prod_{i=1}^{i=k} S_i = \text{SURV}(k)$$

we obtain

$$\frac{\partial V}{\partial L_k} = -\text{SURV}(k) \left[\sum_{i=k}^{i=n} \left\{ \begin{matrix} j=i \\ \pi \\ j=k+1 \end{matrix} S_j \right\} v_i \right] \frac{\partial \alpha_k}{\partial L_k}$$

The term in the square bracket is the estimated value of the remainder of the mission, assuming that the aircraft arrives successfully at the point k . (This is called VALON(k) in the module.) Since α_k is the total attrition for the k th leg, the quantity $\partial \alpha_k / \partial L_k$ is simply the difference between high-altitude and low-altitude rates per nautical mile. Moreover, since we are assuming a constant ratio HILOAT between high-altitude and low-altitude attrition rates, this quantity is proportional to the attrition rate. Therefore, we can write:

$$\frac{\partial V}{\partial L_k} = -\text{SURV}(k) * \text{VALON}(k) + (\text{Attrition Rate}(k)) * \text{CONSTANT}$$

Thus the leg where additional low-altitude range will do the most good can be selected by comparing the product of the first three factors in the above expression for $\partial V / \partial L_k$.

This is the technique used in determining whether the next increment of low-altitude range is to go into the precorridor legs, the leg to the first target, or in extending the low-altitude flight to additional legs or fractions thereof beyond the first target.*

The attrition rate used in this decision process for legs beyond the first target is simply $\text{ATLEG}(k) / \text{DISTLEG}(k)$; thus the effective attri-

* Actually the values of the SURV used in the subroutine during the allocation of the low-altitude flight are all divided by the value of SURV to the first target. This speeds up the operation of the routine, since changes in the survival probability in the precorridor legs or on the way to the first targets, as allocations are made to these legs, do not affect the value of SURV which must be used in later legs.

tion rate also reflects any local attrition associated with the k^{th} route point.

The assumed position-dependent attrition rate per nautical mile is used on the leg to target one so that low-altitude range is added to this leg only as far ahead of the target as is justified by the assumed attrition rate.

The attrition rate used in the precorridor legs is the constant value specified in the data base.

It is also worth noting that regardless of which leg k receives the final allocation of low altitude, this allocation will correspond to some value for the quantity $\partial V / \partial L_k$. This value, of course, is the marginal value of additional low-altitude range. It can be converted (using the conversion factor RANGED) to obtain a marginal value of additional range or the marginal value of saving distance, known as VALDIST, is computed by program POSTALOC and used to estimate the value of the distance saved in alternative sortie definitions.

The above allocation procedure produces a rigorously optimum allocation of the low-altitude range to the sortie so long as there is no local attrition. However, where local attrition is present at specific targets late in the sortie, a theoretically optimum allocation might allocate limited low-altitude range explicitly for each such target. If this were permitted, it could lead to sorties which unrealistically go low for each defended target and fly high between such targets. To avoid this difficulty, the requirement has been imposed that after passing the corridor origin a flight is allowed to go low only once.

Moreover, for simplicity of computation during the development of the sortie definitions, the flight is required to go low before the first target, if it is going to fly low at all. Obviously, if there is local attrition at a target toward the end of the mission but not at the first target, it might be better to stay high past the first target and save the low-altitude capability to be used in the vicinity of later defended targets. While this possibility is ignored (for computational speed) during the development of the sortie definitions, after the sortie definition is complete a final check is made and, if such a change would increase the estimated value of VALSORTY, the change is incorporated in the final version of the flight plan.

If there are no defended targets where the bomber is scheduled to fly high after using its low-altitude range, no changes in the sortie are considered. Otherwise, QUICK tries extending the low-altitude range to include the next defended target. When any low-altitude capability is left prior to the first target of the sortie, the excess is allocated as before between the leg to the first target and the precorridor legs. If there is no such excess, the point where the aircraft first goes low is set as soon after the first target as possible. The resulting value of VALSORTY is then computed. If the sortie value is increased over

that previously obtained, the revised sortie is used. If not, the prior version is retained. This process is repeated until a version of the sortie is tested in which the low-altitude flight is extended to the last defended target. That version of the sortie which produced the best value of VALSORTY is then selected. There is a possibility that in the original version of a given sortie, the total range may be inadequate to execute the sortie as defined, even if the entire mission were carried out at high altitude. In this case, low altitude is not assigned to any of the legs. Moreover, VALSORTY is computed so that it receives no contribution from any route point beyond the maximum range of the aircraft. In this case, later operations usually result in the omission of some targets that cannot be reached or the elimination of recovery, so that a revised sortie definition is developed which constitutes a feasible sortie.

Depenetration Routine: Each bomber for which a recovery is planned must exit via a depenetration corridor. These corridors, while having no attrition associated with them, serve to define the geographic route to be flown while leaving enemy territory. When a bomber leaves a depenetration corridor, it recovers at a base which is associated with that corridor. The bomber chooses the depenetration corridor according to the last target struck in the sortie. If D1 is the distance from that target to the depenetration point, and D2 is the distance from depenetration point to the nearest recovery base associated with that point (or corridor), then the depenetration corridor used is the corridor which minimizes

$$(2*D1) + D2$$

Sortie Modifications: All decisions on the modifications of the sortie definition are based on the estimated effect the changes will produce in the value of VALSORTY.

The initial sortie definition may not even be feasible. It may require too many warheads; it may require too much range; or it may specify all bombs whereas the aircraft may carry ASMs. Thus, the task of program POSTALOC is to revise the sortie definition to produce a feasible sortie with the highest possible expected value of VALSORTY.

In accomplishing this, the program estimates the marginal value of using bombs in a sortie and the potential advantage of using ASMs instead, performing one or more of the following functions.

- o Determine which targets assigned bombs should be converted to ASMs when not all ASMs are assigned
- o Determine which remaining bombs are the least value and should be deleted if too many strikes are assigned
- o Determine which route points (recovery or bomb targets) are of negative value to the sortie and should be deleted.

In so doing, it analyzes each route point in succession down to and perhaps including the recovery point. The processing of each route point is handled in two parts. First, the marginal value of the route point as a target for a bomb is evaluated. Then, the value of the same route point is calculated as a potential ASM target, and the marginal value of changing it to an ASM target is estimated. For these computations the recovery point is not included in the evaluation.

When all ASMs have been assigned, there may still be too many strikes for the available warheads. The next step may then be, still excluding the recovery point, to select the least valuable remaining bomb which could be deleted. Finally, the sortie is evaluated again, this time including the recovery point to be sure that all route points including the recovery make a positive contribution to the payoff.

The marginal value of each route point is also evaluated. The value of reaching the route point, multiplied by the probability of surviving to reach it, is compared with the cost of doing so.

This cost consists of two elements:

- o Change in the probability of reaching succeeding targets because of local attrition, if any, at this target, or because of additional area attrition over the added distance required to fly to this target
- o Reduction in the amount of low-altitude flight available because of the extra distance to the target, which in turn can effect penetration probability to all targets.

In analyzing each target, the program considers an alternative flight route which bypasses the target and goes directly from the preceding to the succeeding target. The effect of this route on the expected payoff for succeeding route points can be directly evaluated. The change in attrition is known, so the change in the cumulative survival probability SURV to the succeeding target can be computed, and the value VALON of the remainder of the sortie is made available.

The change ΔV in VALSORTY, due to change in available low-altitude capability, is only estimated. The estimate is based on the amount of distance saved by skipping the target DISTSV multiplied by the quantity VALDIST, the marginal value of distance. However, where the saving in distance is very large, this type of linear extrapolation with a constant VALDIST can be quite misleading and could even exceed the full value of all targets in the sortie. Obviously, the value of the sortie can never exceed the actual value VALMAX of all route points, and with one target k omitted could not exceed VALMAX-V(k). Consequently, the value VALO of omitting a target k cannot exceed POTVALO-VALMAX-V(k)-VALSORTY. This quantity POTVALO is therefore used to establish a limiting value for the value of saving distance. The quantity VALDIST is used to give the derivative for small values of DISTSV. The actual form used

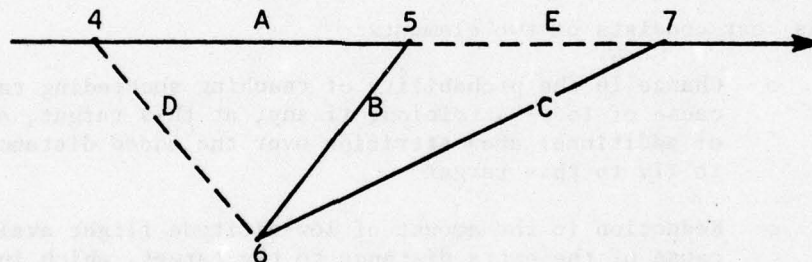
for estimating ΔV for distance saved is:

$$\Delta V = \text{POTVALO} * [1.0 - 1.0/(1.0 + \text{TEMP})]$$

where $\text{TEMP} = \text{VALDIST} * \text{DISTSV}/\text{POTVALO}$.

In the second phase of the process -- to estimate the value of the target as an ASM target -- the time premium for using an ASM on the target is added into the basic value RVAL of the target, and the survival probability used is that for the earliest possible launch point in range of the target.

Determining the value of omitting a route point requires calculation of the distance saved. Once this information has been computed for two successive route points, the next computations are distances that are necessary to determine whether the two points are out of order on the route. The following figure illustrates the method used.



This figure illustrates a route:

- 4 via leg A to 5
- 5 via leg B to 6
- 6 via leg C to 7.

Consider the possibility of reversing the order of points 5 and 6 on the route. The present distance is $A + B + C$; the revised distance would be $D + B + E$, using dashed alternative legs D and E.

If the reversed path is shorter, then $D + B + E < A + B + C$ or $A + C - D - E > 0$. When we consider omitting 5, we compute $\text{DISTSV} = A + B - D$. When we consider omitting 6, we compute $\text{DISTSV} = B + C - E$.

Adding the two values of DISTSV and subtracting $2B$ we obtain $A + C - D - E$. Therefore, if this value is positive the two route points are out of order, and the flag JSEQERR is set to indicate one of the two targets for possible temporary omission. Usually the first target is flagged. (The presumption is that a later evaluation will result in the replacement of such a target in its proper position in the sortie.) However, if the first target is also a launching point for ASMs, even temporary omission would be complicated; thus, rather than seek an

alternative launch point for the ASMs, the second target will be flagged instead. If both route points are also ASM launching points no flag is set, and the current order of targets is not changed.

The problem of route points serving double duty as ASM launch points also arises when the marginal value of omitting route points is being estimated. Therefore, after the original value VALO is estimated, a check is made to see if the point is used as an ASM launch point. If so, the value VALO of omitting the point is decremented to reflect changes in the marginal value of the ASM, for which a new and probably inferior launch point must be found. If such an alternative launch point cannot be found, the entire value of the ASM is charged to VALO. Except in the most extreme cases this is sufficient to preclude omission of this target.

If the program (POSTALOC) is to delete a bomb where the same route point is used as a launch point for an ASM, it first seeks an alternative launch point for the ASM. However, if it cannot find one, the ASM is omitted also.

The desirability of using an ASM on one of the omitted targets is also estimated. This can be done either to find a target for an unused ASM or to evaluate the value of substituting an omitted strike point as the target for an ASM already assigned.

Changes in the bomber route are not considered at this point. In this way, the values of changes considered can be evaluated exactly.

The operation is divided into two portions. First, the program scans all targets in the mission currently assigned for ASMs, skipping any target used as its own launch point, since its omission would change the bomber route. The marginal value of the others is determined by multiplying the value of the strikes as ASM targets by the survival probability for aircraft to the launch point. During this phase, the strike JDEL with the lowest marginal ASM value MINDA is determined.

In the second portion of the operation, all omitted strikes are evaluated as ASM targets. The method of evaluation is exactly the same, except that a suitable launch point must be found. The first route point within range of each target is taken as the potential launch point. As it proceeds through this part of the program, it keeps a record of the strike JADDA with the highest marginal ASM payoff MAXDA and the associated launch point IAIM. Of course, strikes are disqualified for such consideration if another strike on the same target is already in the sortie definition.

The module (POSTALOC) also estimates the value of strikes in the omit list as potential targets for bombs. It does this by finding an additional target or an omitted target that is more profitable for a bomb than the least valuable in the sortie. In turn, each target in the omit list is processed. Each potential target is tried first in a

position just before the first target with a higher value of RHO. The distance added to the sortie is then evaluated. The target is then tried in a position on the other side of its nearest neighbor (nearest in value of RHO). If this position produces a lower value for the distance added, this position is accepted instead of the original position.

The marginal contribution of the bomb in the preferred position is then computed. The method parallels the calculation of the marginal value of bombs in EVALB. The effect of the extra attrition on following targets is evaluated. Then the effect on low-altitude range is estimated using $(VALDIST * DISTAD)$. These quantities are added to get the total benefit VALO of not flying to this new route point. The value of the target, times the probability of surviving to reach it, is then computed to get the net marginal value of adding the target DVALB.

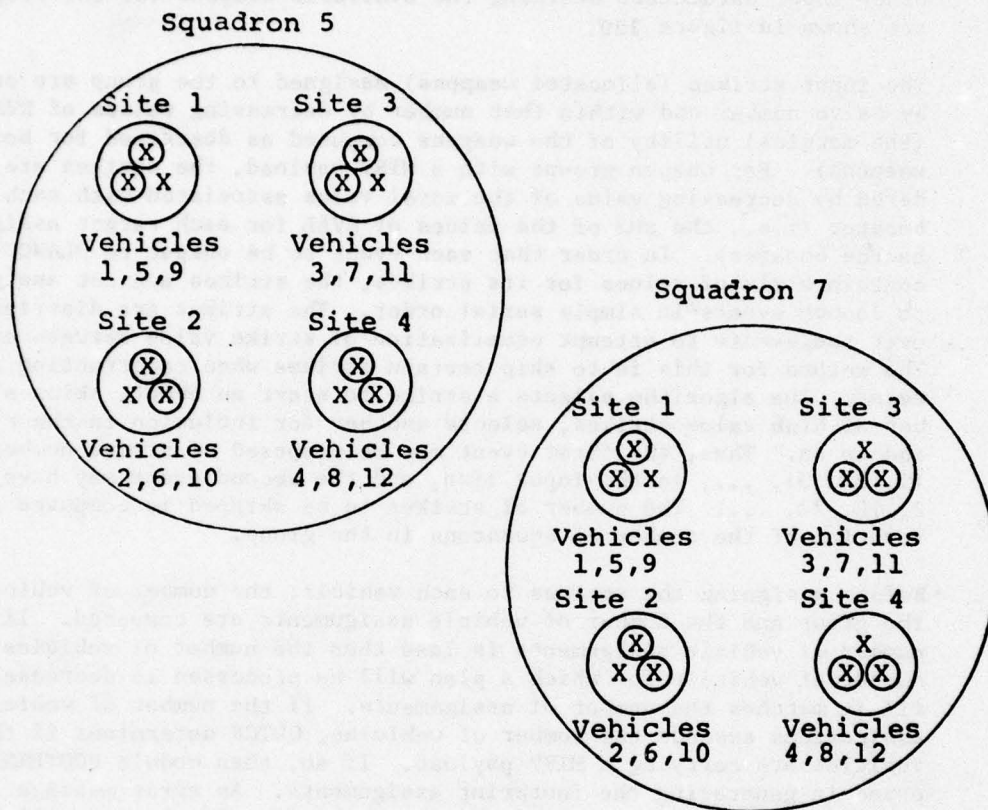
The index for the target with the highest DVALB is then recorded as JADDB, and the route point it should follow is recorded as JAF. Of course, any strike on a target already in the sortie is excluded from consideration to avoid duplicate strikes on the same target by the same bomber.

Missile Plans

Module FOOTPRNT generates missile sortie specifications for each missile weapon group and its assigned targets. Since MIRV missiles are a special case of missiles, the description of the additional processing for these weapons is deferred to the next section. For non-MIRV missiles, individual targets are assigned to individual vehicles. For MIRV missiles, an ordered set of targets is assigned to each vehicle. From these data, specific strikes are assigned to specific delivery vehicles within the weapon group. The development of the missile plans is relatively straightforward. With the exception of the timing computations (e.g., launch time, performed in PLNTPLAN), the missile plans are complete as output by FOOTPRNT.

Figure 129 illustrates the structure of a typical missile group. The group may include several squadrons (two shown) and a squadron may include several sites (four per squadron shown). Each site may have one or more vehicles (three shown). Vehicles are considered to occupy the same site if they are so close together that they would have to be targeted as a simple target. For example, the Polaris squadron of 16 missiles on one submarine is considered to occupy one site, while the Minuteman squadron of 50 missiles occupies 50 separate sites.

On the other hand, any nonalert missiles in a squadron will constitute a separate weapon group. Since the vehicle indices within a squadron may not start from 1, the starting vehicle index ISTART for each squadron



⊗ Vehicles in Group
 X Vehicles not in Group

NOPERSQN = Total Vehicles in Squadron.
 NBASE = Number of Bases (or Squadrons) in Group.
 NWPSITE = Number of Weapons per Site.
 ISTART = Lowest Vehicle Index in Group for Each Squadron.
 NWPNS = Total Vehicles in Group.

Figure 129. Exemplar Configuration of Missiles in a Group

is supplied as an input to the missile assignment phase. This and the other input parameters defining the available weapons for the program are shown in figure 129.

The input strikes (allocated weapons) assigned to the group are ordered by salvo number and within that number by decreasing values of RVAL (the marginal utility of the weapons computed as described for bomber weapons). For weapon groups with a MIRV payload, the strikes are ordered by decreasing value of the total value associated with each booster (i.e., the sum of the values of RVAL for each target assigned to the booster). In order that each event to be output to PLANOUT will contain a mix of values for its strikes, the strikes are not assigned to launch events in simple serial order. The strikes are distributed over the events to attempt equalization of strike value between events. The method for this is to skip certain strikes when constructing an event. The algorithm selects a strike to start an event, skips a number of high value strikes, selects another for inclusion in the event, and so on. Thus, the first event may be composed of strike numbers 1, 11, 21, 31, ..., in the input list, and the second event may have strikes 2, 12, 22, The number of strikes to be skipped is computed as a function of the number of squadrons in the group.

Before assigning the strikes to each vehicle, the number of vehicles in the group and the number of vehicle assignments are computed. If the number of vehicle assignments is less than the number of vehicles, the number of vehicles for which a plan will be processed is decreased until it matches the number of assignments. If the number of vehicle assignments exceeds the number of vehicles, QUICK determines if the vehicles are carrying a MIRV payload. If so, then module FOOTPRNT has erred in generating the footprint assignments. An error message is printed to this effect and processing proceeds. The result will be the omission of some target sets from the final plan. If the group does not have a MIRV payload, the least valuable assigned targets are removed until the number of targets equals the number of vehicles. However, targets assigned through the fixed assignment capability of module ALOC are not omitted, unless there are more fixed assignments for this group than there are vehicles. In that case (an input error), fixed targets are omitted in order of increasing value (RVAL) until the number of targets matches the number of vehicles. In addition, an error message is printed to this effect.

MIRV Missile Plans

Technological developments in guidance have made possible the introduction of multiple missile warheads on a single missile which can be directed at geographically separate targets. Although the original QUICK General War Gaming System was not designed to accommodate multiple independently targetable reentry vehicles (MIRVs), the introduction of MIRVs into operational weapons made it very desirable to incorporate into the QUICK system the changes required to enable the consideration of these weapons.

A major ramification of the addition of the MIRV capability to the system was the necessity to consider the effect upon the target assignments of "footprint" constraints: that is, constraints on the geographic configuration of targets assigned to a single missile equipped with MIRVs. In order to minimize the amount of system alteration required to introduce the MIRV capability, it was decided not to alter the basic weapon allocation process, but rather to introduce these footprint constraints into the plan generation process subsequent to the initial assignments of targets to weapon groups as effected by module ALOC. Hence, the development of the general strike plan now entails, in order of occurrence, the initial allocation of targets to weapon groups in module ALOC, the refinement of the target point locations for complex area targets and the reordering of the assignments according to weapon group in module ALOCOUT, and the construction of specific booster loads (i.e., the weapon-to-target point assignments to be associated with a single MIRV-capable missile) for each weapon group with a MIRV capability in FOOTPRNT. Once module FOOTPRNT has determined the assignment of targets to booster, this information is passed to module PLANOUT. In that program, the booster load assignments are distributed to the individual boosters in each squadron according to the method discussed previously in the section Basic Sortie Generation (Missile Plans). In the MIRV case, however, the value of the sortie is defined to be the sum of all the marginal utility values (RVAL) for the targets assigned to the booster. Program FOOTPRNT orders the booster load assignment information in order of decreasing values of sortie before passing the information to module PLANOUT.

Throughout this discussion, the term "target point" will refer to a "desired group zero" (DGZ) selected either in module ALOC for simple targets, or in module ALOCOUT for complex and area targets, as the aim point for a single reentry vehicle (RV). Although, depending upon the value of a given target, two or more RVs may be allocated to target points with the same geographic coordinates, these target points will be considered as being distinct in all the succeeding processing of the target assignments.

QUICK's MIRV Platform Representation: QUICK does not simulate trajectory paths; it solves mathematical models. Therefore, for footprinting clarification the MIRV mathematical model must be defined. Accordingly, QUICK views each MIRV platform as an initial energy source and, associated with that source, a rate of energy consumption. The energy source may be given in terms of pounds (lbs), as a fuel supply; velocity, as a momentum measure, or any other measure. By selecting targets as being lead targets (that is, the initial target assignment for a given footprint) QUICK constructs feasible footprints by collecting a subset of targets such that there is sufficient energy on the MIRV platform to traverse the targets. A feasible footprint, then is any collection of targets that may be hit within the energy constraints; no other limitations apply within the QUICK model.

The model requires four sets of equations which define footprint feasibility. (For purposes of discussion, assume fuel as being the energy source.) These sets are equations for:

- a. Determining fuel load available for footprinting
- b. Determining maximum booster range and range extension
- c. Determining fuel consumption per mile of equivalent downrange distance (explained below)
- d. Determining factors for converting crossrange and uprange distances to equivalent downrange distances.

Explanations of each set of equations follows.

- o Fuel Load: This represents the initial source level. It may, and usually is, simply set to a constraint; say 1000 lbs. Or, as in some cases, it equals a second order power series equation dependent on the range to the first target within the footprint being developed.
- o Maximum Booster Range: This equation limits the distance from the launch base to the first target within a potential footprint. If the first target is beyond the range of the booster an attempt will be made to use some of the "maneuvering fuel" to reach it. If range extension equations exist they will be used, otherwise the fuel consumption equations for a fully loaded platform will be used.
- o Fuel Consumptions: This set of equations relates the rate of fuel usage on an (equivalent) downrange distance basis. Downrange distances are measured along an axis which is parallel to the shorter of the two great circle routes from the launch point to the first target point to be hit; crossrange distances are measured along an axis which is perpendicular to this route. The uprange direction is defined to be parallel but oppositely directed to the downrange direction. All MIRV systems, to date, give fuel expenditures based on the number of reentry vehicles currently onboard. In terms of footprint construction, this implies one set of equations for reentry vehicle deployment at the first target; another set of equations for the second reentry vehicle deployment at the second target and so on.
- o Crossrange and Uprange Factors: The fuel consumption rates, given above, assume expenditures over a downrange distance. To compensate for target spacing other than downrange, use is made of the equivalent downrange distance (EDD) of a target. The major premise of this method is that all downrange, crossrange, and uprange distances can be converted into an equivalent downrange distance, EDD. The EDD is equal to the downrange

distance that could be traversed by the payload if the same amount of energy were expended as would be required to traverse the distance under consideration. In practice, the EDD from point i to point j, may be expressed by the following relationship:

$$(EDD_{ij})^2 = \begin{cases} DOWN_{ij}^2 + \left(\frac{DR}{CR}\right)^2 (CR_{ij})^2 & \text{if } j \text{ is downrange of } i \\ \left(\frac{DR}{UR}\right)^2 UR_{ij}^2 + \left(\frac{DR}{CR}\right)^2 (CR_{ij})^2 & \text{if } j \text{ is uprange of } i \end{cases}$$

where:

$\frac{DR}{CR}$ = downrange-crossrange ratio as user supplied

$\frac{DR}{UR}$ = downrange-uprange ratio as user supplied

$DOWN_{ij}$ = downrange distance from i to j

CR_{ij} = crossrange distance from i to j

UR_{ij} = uprange distance from i to j

Preliminary Calculations: FOOTPRNT operates at a weapon group level. That is, individual footprints are formulated using weapon strikes from a weapon group. No one footprint may contain strikes from differing weapon groups. To minimize the chance that the elimination from booster assignments of targets which will not fit into a feasible footprint will cause an underutilization of the available weapon stockpile, additional RVs are created for MIRV weapon groups in module PREPALOC for assignment in module ALOC. However, after the processing in module FOOTPRNT, the expected number of RVs actually utilized will not exceed the number that are available.

Accordingly, individual strikes associated with a weapon group are read and stored and, then, sorted according to azimuth (relative to the weapon launch base). This sort aids in strike selection since all observable MIRV platforms to date have a severe penalty of traversing cross-range relative to downrange or uprange targets with similar azimuths have a greater potential of forming feasible footprints.

Strike Selection: From the group weapon strike list, a subset is collected that potentially may form a viable footprint.

The collection process queries the sorted strike list and nominates strikes that are unassigned to boosters as being the first target (called lead) to be hit within the footprint being formed. Given the lead target, testing continues within the sorted strike list but now additional strikes are added only if they fall within a geographic area relative to the lead target. The geographic area is the contour of a maximum footprint as defined by the MIRV platform equations. If strikes fall within this area there is a high probability that the collection will form a feasible footprint. Neighboring strikes are collected until a sufficient number are collected. This normally equals the onboard loading factor plus 50 percent (an arbitrary value) of that number. This collection is now tested for footprint feasibility.

Booster Assignment: The subset list of collected strikes are now interrogated in order to find an acceptable "path" if it exists, which when traversed will hit the proper number of targets within the fuel constraints of the MIRV platform; hence forming a footprint.

A dynamic programming algorithm (plus branch and bound solutions, appendix B) determines paths. This solution is a variation of the classical traveling salesman problem which requires a salesman to visit N cities once and only once in a path such that he visits the cities with minimal total distances traveled. For the footprint problem cities become target assignments and distance becomes the fuel costs of going from assignment I to J.

The solution consists of working the problem backward. Starting at the end condition (one RV onboard) the minimal path of arriving there is found by querying the costs of all potential second stage targets. Once found, the second stage (two TVs onboard) is processed to find the optimal I,J path of arriving there with three RVs onboard. This method continues until each I,J minimal combination is obtained for all onboard stages. The global optimal is a summation of local optimals of going from one stage to the next.

Upon processing all stages, a feasible path is defined if there is sufficient fuel and no target is contained within the path more than once. If a feasible path is obtained, strikes are assigned to a booster. For the case of repeating assignments, branch and bound algorithms are employed (see appendix B).

Other strike subsets are now formed and processing continues.

A.2 Detailed Sortie Specifications

Module PLANOUT processes the bomber and missile plans prepared by modules FOOTPRINT and POSTALOC and writes them with tanker plans in a format required by processors external to QUICK.

Table 12 indicates the type of sortie information required by PLANOUT for each sortie. Besides sortie identification, launch base, and

Table 12. List of Information Required by PLANOUT

<u>CATEGORY</u>	<u>ITEM</u>	<u>ITEM</u>
Sortie Identification		Group index Corridor index Sortie index
Base Information		Base index Base location (lat., long) Regional index Payload index Weapon type
Vehicle Information		Vehicle index Vehicle speed (at high and low altitude) Vehicle range (with and without refueling)
Sortie Information		Refuel index Depenetration corridor Alert status Delay before takeoff Number of targets Target list Low-altitude range available

vehicle information, it describes the target area part of the sortie by listing the target events. It lists the targets to be attacked, their location, and index numbers. It also lists ASM targets, decoy launches, and whether the bomber recovers or aborts the mission.

The major functions performed by PLANOUT in processing the input sortie data and creating the detailed sortie specifications include: assigning refuel areas to bombers and allocating tankers to service them; calculating ASM launch points, change altitude, and launch decoy events should occur; coordinating launch times according to user parameters; and calculating distances and times between all events of each plan. The techniques associated with each of these functions are discussed below.

Bomber Plans

Figure 130 shows a typical flight route for a long-range bomber sortie from launch to recovery. After launching from its base, the bomber flies first to a refuel area if refueling is called for, then to a corridor entry point. It may then fly one or more prespecified doglegs (called corridor legs) which define a penetration route before reaching the point labelled corridor origin. From the origin, it flies over a target area and its assigned targets in their proper order. Finally, it enters the depenetration corridor, which may also consist of one or more doglegs, before going on to the recovery base. Depending on the bomber's range, a portion of the flight route may be flown at low altitude.

Module PLANOUT generates a detailed sortie plan for each bomber which defines the flight route, altitude profile, and offensive operations. The major PLANOUT functions and techniques involved in preparing the detailed bomber sortie data are discussed in the following paragraphs.

Distance Calculations: Each event of the bomber sortie is assigned a place of occurrence in latitude and longitude. Later, the great circle distances between all events are computed in nautical miles. The standard law of cosines for a spherical triangle is applied to compute the great circle distance. The radius of the earth is assumed to be 3440.068 nautical miles.

Bomber Timing: Using the calculated distances between events together with bomber (high or low altitude) speed or ASM speed, the time intervals between successive bomber events are computed. For the purposes of QUICK, each event of a plan is assumed to be carried out instantaneously at a specified time; i.e., a bomber is assumed to be launched in no time, to refuel without delay, and to change altitudes instantly.

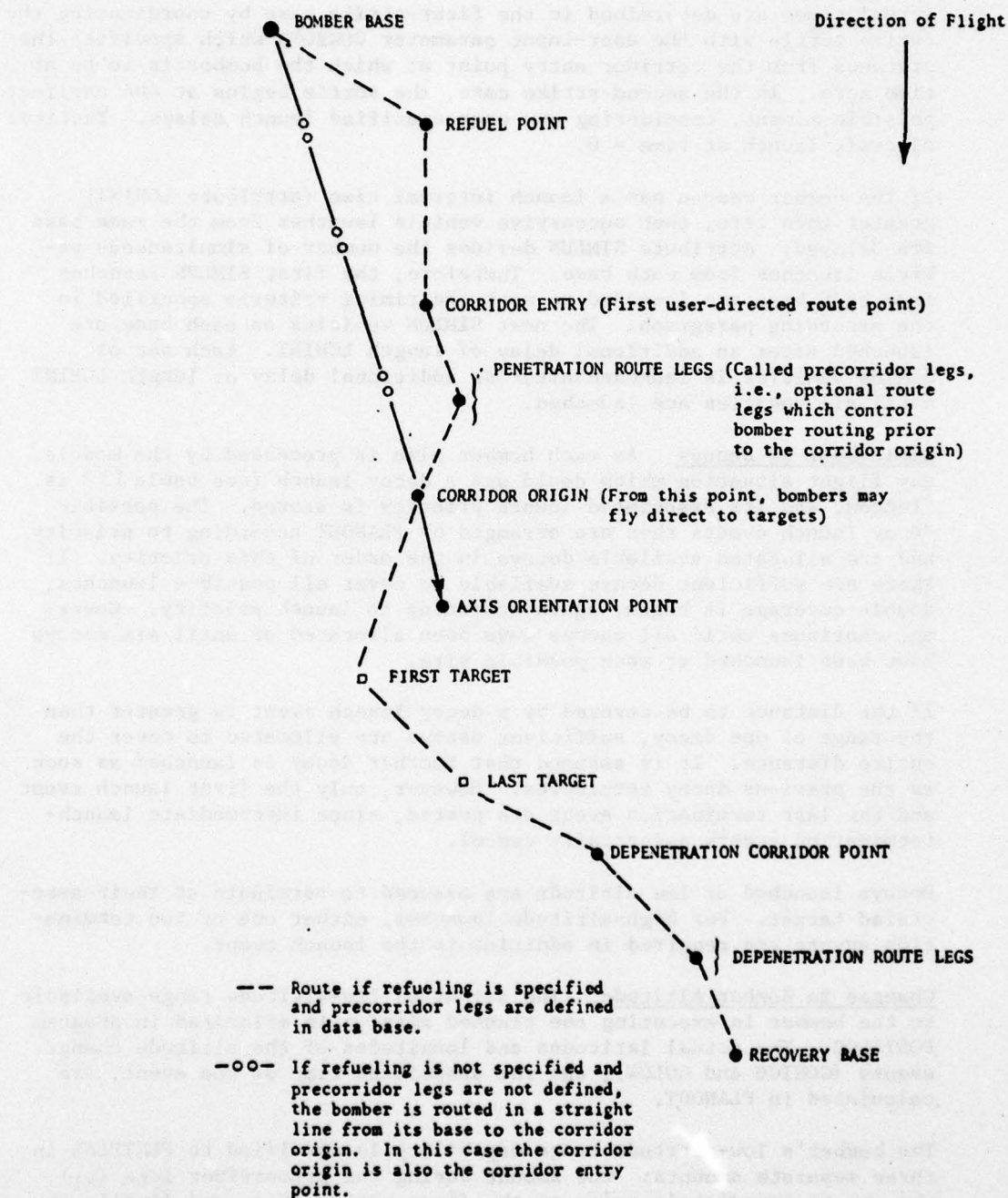


Figure 130. Typical Bomber Flight Route

Actual times are determined in the first-strike case by coordinating the entire sortie with the user-input parameter CORBOMB which specifies the distance from the corridor entry point at which the bomber is to be at time zero. In the second-strike case, the sortie begins at the earliest possible moment, considering any user-specified launch delays. Tactical aircraft launch at time = 0.

If the bomber weapon has a launch interval time (attribute LCHINT) greater than zero, then successive vehicle launches from the same base are delayed. Attribute SIMLUN defines the number of simultaneous vehicle launches from each base. Therefore, the first SIMLUN launches from each base are launched to meet the timing criteria specified in the preceding paragraph. The next SIMLUN vehicles on each base are launched after an additional delay of length LCHINT. Each set of SIMLUN vehicles is launched after an additional delay of length LCHINT until all vehicles are launched.

Employment of Decoys: As each bomber plan is processed by the module, any flight situation which could use a decoy launch (see table 13) is flagged, and its associated launch priority is stored. The possible decoy launch events then are arranged by PLANOUT according to priority and are allocated available decoys in the order of this priority. If there are sufficient decoys available to cover all possible launches, double coverage is begun, again according to launch priority. Coverage continues until all decoys have been allocated or until six decoys have been launched at each possible site.

If the distance to be covered by a decoy launch event is greater than the range of one decoy, sufficient decoys are allocated to cover the entire distance. It is assumed that another decoy is launched as soon as the previous decoy terminates. However, only the first launch event and the last termination event are posted, since intermediate launch-termination events essentially cancel.

Decoys launched at low altitude are assumed to terminate at their associated target. For high-altitude launches, either one or two termination events are required in addition to the launch event.

Changes in Bomber Altitude: The amount of low-altitude range available to the bomber in executing the planned mission is allocated in program POSTALOC. The actual latitudes and longitudes of the altitude change events (GOHIGH and GOLOW), and the associated time of the event, are calculated in PLANOUT.

The bomber's low-altitude range capability is specified to PLNTPLAN in three separate amounts: the amount during the precorridor legs (G_1), the amount immediately prior to the first target (G_2), and finally the amount following the first* target (G_3). For realism, values of G_1 , G_2 , G_3 equivalent to less than 15 minutes are not applied.

* If the value of attribute PAYALT is HIGH, this low altitude range consists of the distance between the last target and recovery.

Table 13. Launch Priority

<u>LAUNCH PRIORITY</u>	<u>CIRCUMSTANCES OF LAUNCH</u>
1	[*] R _L miles before first low-altitude gravity bomb attack on a SAM-defended target
2	Immediately before changing from high to low altitude
3	Immediately before penetrating defended airspace if flying at high altitude
4	^{**} R _H miles before first high-altitude gravity bomb attack on a SAM-defended target
5	Coverage when flying at high altitude over defended airspace before priority 4 launch
6	R _L miles before subsequent low-altitude gravity bomb attacks on SAM-defended targets
7-8 ^{***}	Coverage when flying at high altitude over defended airspace after priority 4 launch

^{*} R_L = range of decoy at low altitude

^{**} R_H = range of decoy at high altitude

^{***} Priority 8 is used if the coverage is to begin at the point where the priority 4 decoy terminates. Priority 7 is used if the bomber has changed altitude between the priority 4 and the priority 7 launch.

G_1 is measured backward from the corridor origin toward the corridor entry points. Since corridor attrition may or may not be associated with the precorridor legs, the low-altitude range capability is applied against only those precorridor legs where the bomber would experience attrition. Any G_1 remaining is added to G_2 .

The initial go-low point after the precorridor legs is determined from the value of G_2 :

- a. If $G_2 > 0$, the go-low event will occur G_2 miles before the first target. Here, the first target is defined to mean the first bomb target on the first ASM launch point after the corridor origin.
- b. For plans in which $G_2 = 0$, the bomber will go-low at the first target, provided that the range to be flown at low altitude after the first target (G_3) > 0 . If G_3 also equals 0, it will fly the entire mission after the corridor origin at high altitude.
- c. If $G_3 < 0$, the bomber will fly $-G_2$ miles beyond the first target before going low; the total low-altitude range in this case is $G_3 - (-G_2) + G_1$ miles.

G_3 is always measured out beginning at the first target,* and any G_3 remaining after the target area is applied during depenetration.

The location of the change-altitude points are initially determined by applying G_1 , G_2 , and G_3 as outlined above. Once the initial processing is completed, the GOHIGH and GOLOW locations are checked to ensure that the bomber does not change altitude in an unrealistic manner. If necessary, as explained below, the location of these points is modified.

For the purposes of the QUICK system, each event of a plan is assumed to be carried out instantaneously at the indicated time; i.e., a bomber is assumed to be launched in zero time, to refuel with no delay, and to change altitude instantaneously. Thus, if the bomber is asked to go high or go low in the immediate neighborhood of a target or ASM launch point, the order of these events does not matter. However, the detailed plan appears more realistic if the bomber climbs immediately after, rather than immediately before, a target and goes to a low altitude immediately before, rather than immediately after, a target.

Module PLANOUT adjusts the plan to make certain that this is the case. The adjustment performed is seen by referring to figure 131 where the high-altitude adjustment is shown. If a bomber is found to climb within THB minutes before a target (in which case it might be thought of as flying a path shown by the solid line in the figure), then the altitude

* However, if the value of attribute PAYALT is HIGH, G_3 is measured out beginning at the last target.

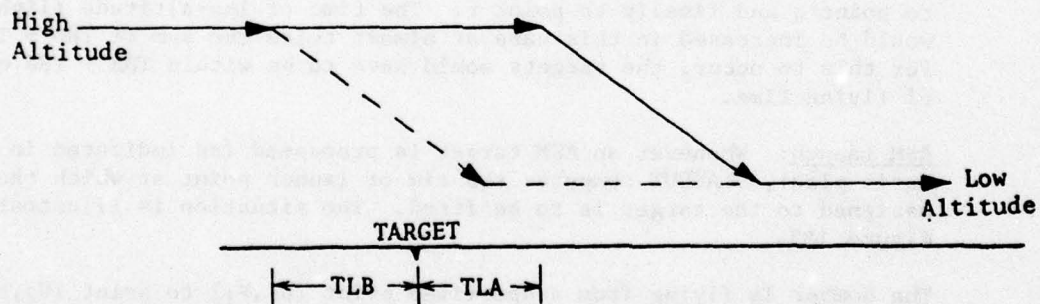
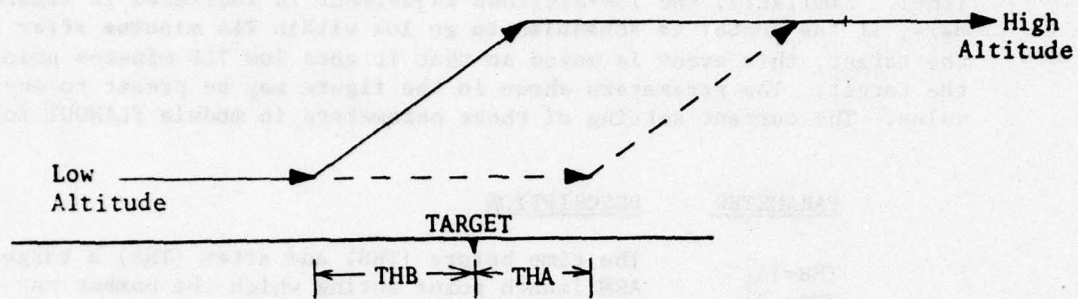


Figure 131. High-Altitude and Low-Altitude Adjustment

change event is moved so that it occurs THA minutes after the target (in which case it might be thought of as flying the path shown by a dotted line). Similarly, the low-altitude adjustment is indicated in figure 131. Here, if the bomber is scheduled to go low within TLA minutes after the target, this event is moved so that it goes low TLB minutes prior to the target. The parameters shown in the figure may be preset to any value. The current setting of these parameters in module PLANOUT follow.

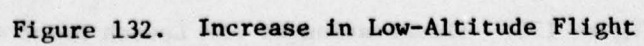
<u>PARAMETER</u>	<u>DESCRIPTION</u>
THB=15 } THA= 2 }	The time before (THB) and after (THA) a target or ASM launch point during which the bomber may not change from low to high altitude
TLB=10 } TLA= 3 }	The time before (TLB) and after (TLA) a target or ASM launch point during which the bomber may not change from high to low altitude

In making these adjustments, the amount of low-altitude flight is never decreased, but it may be increased as illustrated in figure 132. It shows two targets labeled T₁ and T₂ with associated values of the parameters THB and THA. A section of bomber path is shown by dashed lines. In this case, a GOHIGH event found, say, at point p would be moved first to point q and finally to point r. The time of low-altitude flight would be increased in this case at almost twice the sum of THB + THA. For this to occur, the targets would have to be within THB + THA minutes of flying time.

ASM Launch: Whenever an ASM target is processed (as indicated in the basic plan), PLANOUT computes the aim or launch point at which the ASM assigned to the target is to be fired. The situation is illustrated in figure 133.

The bomber is flying from a specified point (U₁,V₁) to point (U₂,V₂) and is to fire an ASM at a target (UAT,VAT) enroute, at maximum range R if possible. The aim point to be determined is (RLAT,RLONG). In determining the point (RLAT,RLONG), two cases occur:

- a. For simpler case exists when the range of the ASM is sufficient for it to be launched while the bomber is proceeding in a straight-line path from point (U₁,V₁) to (U₂,V₂). This would be the case if the range of the ASM were R' (figure 133). The ASM target is then said to be "in range." Since it could be launched at maximum range from either point p or p' shown in the figure, the point p would be chosen as the desired launch point. Since point p is a point enroute, it is not considered to be a flypoint.



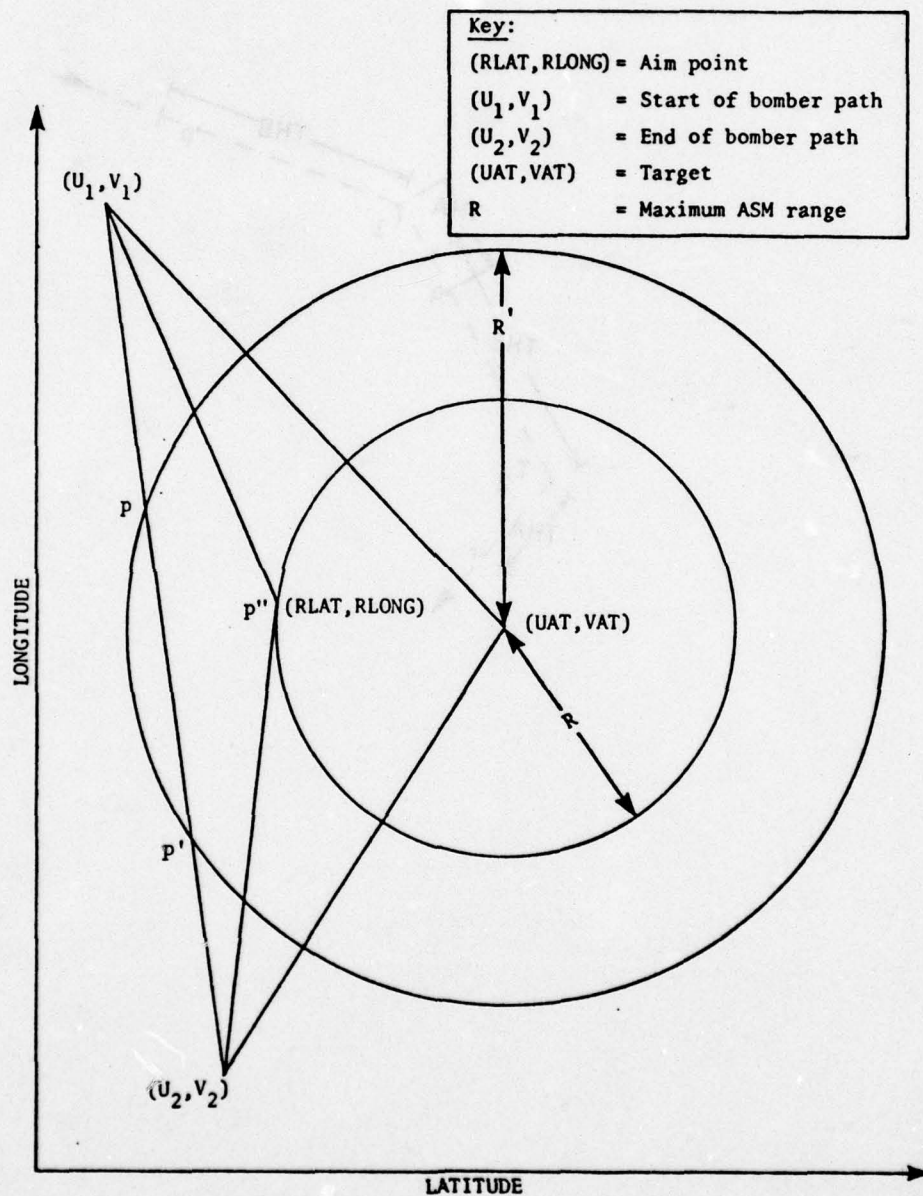


Figure 133. Illustration of ASM Launch Point Calculation

- b. The more interesting case occurs when the range of the ASM is equal to R in figure 133. Here, the bomber must deviate from its course and fly to the point p" to fire the ASM. The ASM target is noted as "out of range," and the point p" is now a flypoint.

Subsequently, during allocation of low-altitude range, any ASM launch scheduled to occur at the corridor origin will be rescheduled to occur 5 minutes later if the aircraft is also to change to a low altitude at the origin.

Bomber Refueling

The QUICK design provides for modeling two kinds of bomber refueling capabilities: "buddy" and area. In buddy refueling, two aircraft take off together and fly to the refuel point; one then provides fuel to the second and recovers. Fuel can be provided by either a tanker or another bomber of the same squadron as the aircraft being refueled.

There are two types of area refueling: directed and automatic refueling. In the directed mode, the user establishes, in the data base, a specific refueling area (up to 20 per side may be defined in the data base) and manually assigns the appropriate bombers and tankers to this area. In the automatic mode, the Plan Generator (module PLANOUT) develops the refueling plan on the basis of information provided in the data base. The data base reflects the bomber squadrons which require refueling and tankers which are available. Module PLANOUT then selects the refueling area (up to 30 additional refueling areas may be added) and assigns the bombers and tankers accordingly. To reflect the refueling requirements associated with a specific plan, the user defines the attribute IREFUEL for all bomber and tanker units defined in the data base. The codes which may be assigned as the value of IREFUEL are as follows:

<u>IREFUEL SETTING</u>	<u>DEFINITION</u>
-5	Automatic refueling -- two refuelings required
-4	Automatic refueling -- one refueling required
-3	This code is used to flag air-breathing missiles which are to be treated as aircraft when calculating attrition rates -- no refueling involved
-2	Buddy refueling -- a bomber from the same squadron is used in a tanker role

IREFUEL SETTINGDEFINITION

-1

Buddy refueling in which support is provided by a tanker. Tanker units associated with buddy refueling need not be defined in the data base

0

No refueling required

≥1

Directed area refueling -- refuel area and bomber/tanker assignments are directed by user

For the weapon allocation process to reflect accurately the appropriate range of all available aircraft, it is necessary to decide prior to the allocation which aircraft have their refueled range and which do not. If the user has specifically assigned the refuel area and/or buddy refueling capabilities, the program assumes that the aircraft can be refueled and so indicates to the weapon allocation portion of the program. If the user selects the automatic refueling capability, there may not be enough tankers, and therefore a decision must be made in module PLANSET as to which bombers are to be refueled and which are not. If a count of the bombers requiring automatic refueling and the tankers available to perform this refueling indicates a deficiency of tankers, the aircraft are given the refueled range on the basis of a set of priorities built into the program. Alert aircraft are always given priority over non-alert; aircraft with the least unrefueled range are given priority over those with a greater range. Thus, when the weapons are allocated, the range capability has been completely determined, and the sorties generated by module POSTALOC assume either the refueled or unrefueled range generated by module PLANSET. Where bombers are used as tankers in buddy refueling (i.e., a bomber unit is assigned the refuel index IREFUEL=-2), the number of bombers available for the strike is cut in half.

Selection of Refueling Areas: For the directed area mode of refueling, the user assigns refuel areas for both bombers and tankers, and the vehicles are scheduled accordingly. Where buddy refueling is to occur, tankers are ignored by the system. Bombers are scheduled to refuel at the "buddy point," which is at maximum range (as defined below) or at the corridor entry, whichever is earlier. The maximum range is determined by:

- a. Let REFDIF = the bomber's refueled range minus range.
Let DIS = the distance (in nautical miles) from base to corridor entry.
- b. If $DIS < REFDIF$, let FACTOR = the greater of $\frac{DIS-5}{DIS}$ or zero.
If $DIS > REFDIF$, let FACTOR = $\frac{REFDIF}{DIS}$.

- c. Now using FACTOR, the desired point is found by an interpolation along the great circle route between launch base and penetrated corridor entry point if the longitudinal difference between base and entry point is greater than 2.8 degrees. Otherwise, the desired point is determined, using a straight line or Mercator interpolation.

For the third case, in which a bomber is to be automatically assigned a refuel area by PLANOUT, the buddy refuel point X is first computed as for buddy refueling. The list of tanker bases is then scanned to see whether the point X is within range of any of them. If not, the closest tanker base is chosen, and a new buddy point is calculated by interpolation. The new point will fall between the tanker base and the original buddy point and will be within range of the tanker base. Next, the refuel area nearest the buddy point (if one exists within a predetermined radius) is selected. Let REFDIF be the difference between refueled range (see figure 134). If there already exists refuel areas which are within REFDIF of the base and within the specified distance D of the buddy point X, the area nearest X is assigned as the bomber's refuel area. Otherwise, the point X is assigned and added to the list of refuel areas. Available tankers will later be assigned and scheduled by PLANOUT in such a way as to service all automatically assigned bombers.

Recovery: The list of targets for a bomber terminates in either of two ways:

- a. With a DEPEN event, indicating normal recovery to the most distant of the four recovery bases associated with its depenetration point, that the bomber can reach. The depenetration corridor description is obtained from the system's input data, and the bomber's dogleg events, if any, are posted in the proper order. Any remaining low-altitude range (G₃) is applied at this time.
- b. With a LAND event, which indicates the aircraft does not have sufficient range to recover. In this case a GOHIGH event is posted if currently at low altitude and an ABORT event defined when the bomber's range (fuel) is exhausted.

Missile Plans

The input missile plans prepared by FOOTPRNT are complete with the exception of the launch and flight times associated with the mission. These calculations are performed in module PLANOUT, and the appropriate data are added to the basic missile plan.

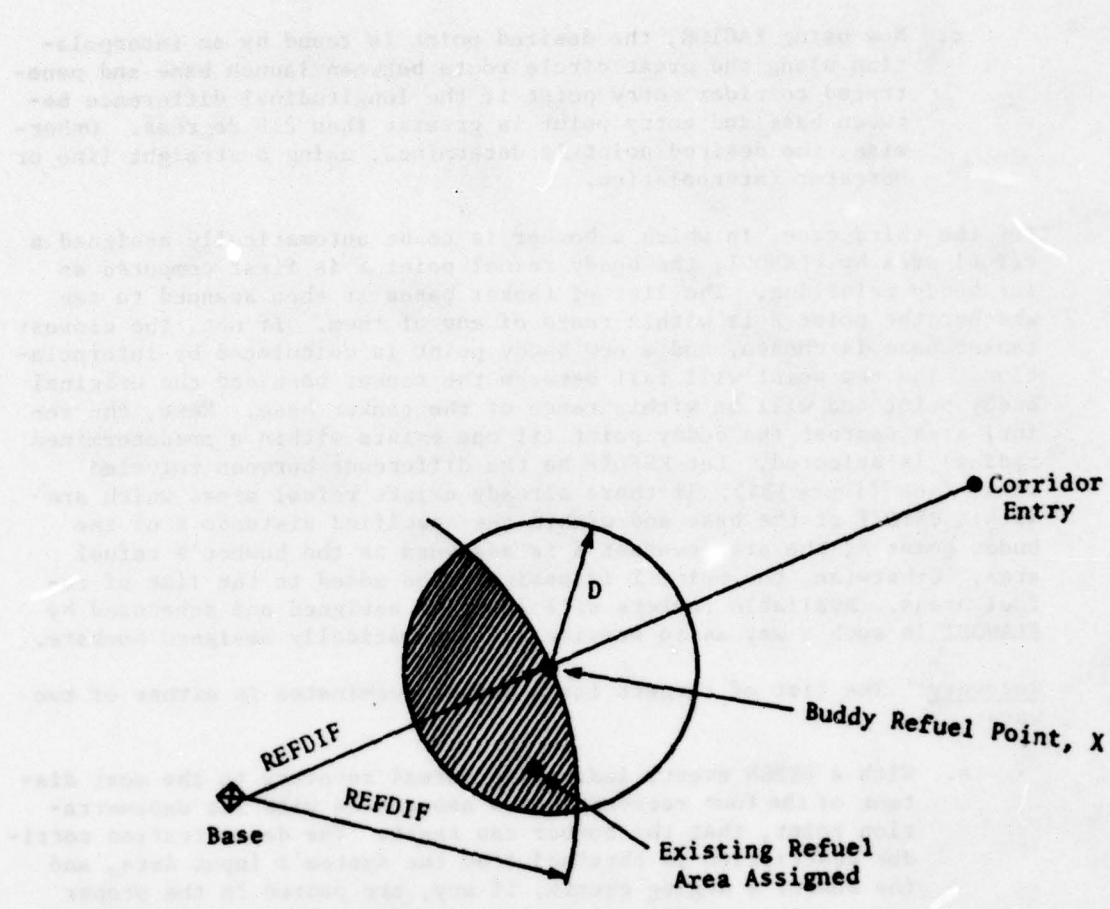


Figure 134. Assigning a Refuel Area (Automatic)

Missile flight times and launch times are calculated from user inputs. The main timing parameters used are the minimum flight time (attribute TOFMIN) and the coordination time for missiles (CORMSL).

Attribute TOFMIN, the minimum flight time for a missile type, may be equal to or greater than zero. All flight times less than TOFMIN will be raised to TOFMIN before the launch and impact times are posted to the missile plan.

The user may specify a CORMSL^{*} for each missile type. This parameter will control the launch timing for initial strikes (INITST=1). There are two kinds of CORMSL: a "FLIGHT" CORMSL and a "LINE" CORMSL.

A "FLIGHT" CORMSL is the fraction of the missile's flight which is completed at time 0.0 exclusive of launch delay. Clearly, such a CORMSL must lie between 0 and 1. If it is 0, the missile is launched at time = 0 plus a launch delay. If it is 1.0, the missile impacts at time = 0 plus a launch delay.

The "LINE" CORMSL requires another user input. The user first specifies a sequence of straight-line segments (not necessarily connected). The "LINE" CORMSL is then the time at which the missile first crosses any line. If the flight path does not cross any line, then the missile will impact at time = 0 plus a launch delay. Because of the great length of missile paths, great circle routes are used for the lines and the flight paths, rather than a Mercator projection of coordinates. The timing calculations which involve "LINE" CORMSLs are described in section 4.

The launch delay specified in the preceding paragraphs is not the alert or nonalert delays specified in the data base. The launch delay is computed using the launch interval time (attribute LCHINT) and the salvo number selected in module ALOC. If the weapon has a zero value for LCHINT, then the launch delay is zero. If LCHINT is greater than zero, then the launch delay is equal to the produce of LCHINT and the quantity salvo number minus one. In other words, the second salvo follows the first after a delay of length LCHINT, the third salvo follows the second after a delay of LCHINT, etc.

If the missile is a fixed weapon with a specific time of arrival specified from module ALOC, this time is used to compute the launch time, regardless of any CORMSL or launch delay. The CORMSL parameter is also ignored for second-strike plans, but the play delay and launch delay are used.

The launch delay is computed as described in the preceding paragraphs. The plan delay is the alert or nonalert delay specified in the data base for the missile group. The launch time in the second-strike case is equal to the sum of the plan delay and the launch delay.

^{*} In module ALOC, a single value for the parameter CORMSL is applied to all missile types.

In the case of missiles with a MIRV capability, if there are several targets assigned to the missile and more than one has a "fixed time" assigned, only the first fixed-time assignment encountered will be considered. Thus, if a previous fixed-time assignment has determined the launch time for the missile, no further calculations are done to compute the launch time for later reentry vehicles on the missile. If there are no fixed assignments (with timing) on a missile with a MIRV payload, the launch time is computed by considering only the data for the target assigned to the first reentry vehicle on the booster.

Tanker Plans

In addition to defining the basic missile and bomber plans, PLANOUT generates the tanker plans for tankers used in an area refueling mode. The input data for tankers are obtained from the data base and include:

- o Tanker base latitude and longitude
- o An index specifying either the refuel area to which it is to be directed or its availability for automatic allocation by PLANOUT
- o Number of tankers per squadron
- o Number of tankers on alert per squadron
- o Tanker speed (referred to below as V_t)
- o Alert delay
- o Nonalert delay
- o Total time on station (TTOS)
- o Tanker type
- o Tanker range.

After all bomber plans have been formulated, a plan for each tanker is generated consisting of the seven events shown below:

<u>EVENT TYPE</u>	<u>TIME BETWEEN EVENTS</u>	<u>PLACE</u>
Launch	Delay	Tanker base
Enter Refuel Area	$DIST/V_t$	Refuel area
Leave Refuel Area	TTOS	Refuel area
Recover ₁	DI_1/V_t	Nearest recovery base
Recover ₂	DI_2/V_t	2nd nearest recovery base
Recover ₃	DI_3/V_3	3rd nearest recovery base
Recover ₄	DI_4/V_4	4th nearest recovery base

where DIST = Distance from tanker base to refuel area

DI_x = Distance from refuel area to recovery base_x

First, PLANOUT assigns a refuel area to each tanker that is not user-directed to a specific area. This is done in such a way as to minimize the total tanker miles flown while servicing all bomber requests. The time of arrival at the refuel area differs depending on whether the plan is for a first or second strike.

In the second-strike case, all tankers are sent to their assigned refuel areas at the earliest possible moment, considering delays before launch due to alert or nonalert status as well as the travel time required between base and refuel area.

In the first-strike case, each tanker is scheduled to enter its assigned refuel area .1 hour prior to the arrival of the bomber that it is to service. The tanker launch time, then, is computed by:

$$\text{Launch time} = (\text{time due at refuel area}) - \left(\frac{\text{DIST}}{V_t} \right)$$

Each tanker is scheduled to leave the refuel area TTOS (total time on station) hours after arriving. The four recovery bases closest to the refuel area are found, ordered by ascending distance, and posted for each tanker as alternate recovery events. The flight times from refuel area to each recovery base are determined and the tanker plan is complete.

A.3 Actual Height of Burst Calculations

QUICK System plan generation provides files for damage assessment programs and simulators outside the QUICK system (e.g., SIDAC, NEMO). On these files, the actual height of burst of each strike is output by the detailed sortie generation program.

For ground bursts, the actual height of burst is output as 000. For air bursts, the detailed sortie generation program computes the actual air height of burst and outputs that variable in hundreds of feet. Actual air heights of burst range from 100 feet to 99,900 feet.

The actual air height of burst is calculated using two quantities: the target vulnerability (VN) and the weapon yield. The first step in the calculation is the computation of the adjusted vulnerability number (AVN) by methods described in "Computer Computation of Weapon Radius," B-139-61, Air Force Intelligence Center. This computation requires input of the VN and the cube root of the weapon yield. Next, the scaled height of burst (SHOB) is selected as shown in table 14. A P-type target is signified by the letter "P" as the third character of the vulnerability number VN. A Q-type target has the letter "Q" as the third character of VN. Finally, the actual height of burst (AHOB) is calculated by the following equation:

Table 14. Scaled Height of Burst Selection

P-Type Targets

Adjusted Vulnerability AVN	Scaled Height of Burst SHOB (feet)
0-9	900
10-12	800
13-14	700
15-17	600
18-19	500
20-23	400
24-26	300
27-39	200
>29	100

Q-Type Targets

0-10	900
10-11	800
12-13	700
14-14	600
15-26	300
>26	200

$$AHOB = \frac{SHOB * W^{1/3}}{100}$$

where: AHOB = actual height of burst in hundreds of feet (i.e., AHOB = 9 for 900 feet)

SHOB = scaled height of burst in feet

W = weapon yield in kilotons.

A.4 Tanker Allocation Technique

The task of allocating tankers to refuel areas in such a way as to service all bombers is considered by PLANOUT to be a form of the classical transportation problem. The variables involved are considered as follows:

Diagram illustrating a rectangular grid representing a refueling area. The grid is defined by rows i and columns j .

- Columns are labeled $j = 1, 2, 3, \dots, C$.
- Rows are labeled $i = 1, 2, 3, \dots, R$.
- The grid is divided into cells.
- Labels for the grid boundaries:
 - Top: $j = 1, 2, 3, \dots, C$
 - Left: $i = 1, 2, 3, \dots, R$
 - Bottom: $b_1, b_2, b_3, \dots, b_C$
 - Right: $a_1, a_2, a_3, \dots, a_R$
- Definitions:
 - j = Refuel area number
 - i = Tanker base number
 - b_j = Total number of tankers required at refuel area j
 - a_i = Total number of tankers available at tanker base i

Each cell in the above table has two entries associated with it.

- a. $COST(i,j)$ = distance from base i to refuel area j + safety factor of .5 miles.
- b. X_{ij} = number of tankers at base i to be assigned to refuel area j .

The statement of the transportation problem to be solved is:

Given: all i, j, a_i, b_j , and $\text{COST}(i,j)$,

Find: all X_{ij} such that the total number of tanker miles flown

$$\left(\sum_{i=1}^R \sum_{j=1}^C [\text{COST}(i,j) * X_{ij}] \right)$$

is minimized, subject to the constraints that

- a. The total number of tankers assigned from base i must equal the total number of tankers available at base i

$$\sum_{j=1}^C X_{ij} = a_i \quad \text{for } 1 \leq i \leq R$$

- b. The total number of tankers assigned to refuel area j must equal the total number required at refuel area j

$$\sum_{i=1}^R X_{ij} = b_j \quad \text{for } 1 \leq j \leq C$$

A dummy refuel area is created to handle extra tankers, which are later reassigned.

The solution is found using Vogel's Approximation Method. This method will be illustrated below by use of an example: additional information may be found in a basic operations research text, such as Introduction to Operations Research by F.S. Hillier and G.J. Lieberman, published by Holden-Day, Inc.

Figure 135 illustrates the formulation of a tanker allocation problem. There are three refuel areas and three tanker bases. We notice, for example, that there are eight tankers at tanker base 3 and 20 tankers are needed at refuel area 2. The distance from tanker base 1 to refuel area 2 is 200 miles, and the distance from tanker base 3 to refuel area 1 is 500 miles.

We wish now to allocate the tankers from the tanker bases to the refuel areas in such a way that all the tankers at the bases are used, all the requirements at the refuel areas are met, and so that the total mileage that all the tankers fly is as small as possible.

Suppose we look at tanker base 1 and try to allocate the 20 there to the refuel areas. There are many possibilities. We could send five tankers to refuel area 1 and 15 to refuel area 2. We could send all 20 to

Refuel Area Tanker Base	#1 10 Tankers Needed	#2 20 Tankers Needed	#3 10 Tankers Needed
#1 20 Tankers Available	210 Miles	200 Miles	210 Miles
#2 12 Tankers Available	500 Miles	220 Miles	500 Miles
#3 8 Tankers Available	500 Miles	220 Miles	500 Miles

Figure 135. Formulation of a Tanker Allocation Problem

refuel area 2. We could send 10 to refuel area 1 and 10 to refuel area 3. Or we could make many other allocations. Our first impulse would be to send all 20 tankers to refuel area 2 because then each tanker would have to fly only 200 miles for a total of 4,000 miles. If we did this, however, refuel area 2 would be saturated and the tankers from bases 2 and 3 would have to be sent in some order to refuel areas 1 and 3, a distance for each tanker of 500 miles or for all 20 tankers a total distance of 10,000 miles. This allocation, then, of all 40 tankers would give a total mileage of 14,000.

If, however, we started all over again and sent 10 of the tankers on base 1 to area 1, the other 10 tankers on base 1 to area 2, this forces us to send the tankers from bases 2 and 3 on a much longer route.

To be more specific, the penalty for not sending the tankers from base 1 on the shortest route to a refuel area is much smaller than the penalty for not sending the tankers from bases 2 and 3 on the shortest route. The idea is that if the tankers are not sent on the shortest route to a refuel area, they can probably be sent on the next shortest route. Therefore, if the distance along the shortest route is not significantly different from the distance along the next shortest route, there is no great penalty for sending the tanker on the second shortest route.

We formalize this idea by defining for a transportation matrix (as in figure 135), a row penalty, which is the difference between the second shortest distance in each row. For figure 135 the row penalties are 10 miles, 280 miles, and 280 miles for rows 1, 2 and 3. We see immediately from these numbers that the penalty for not allocating tankers from row 1 to the closest area is very small compared with the penalty for not allocating from rows 2 and 3 to the closest area. We would naturally then allocate from rows 2 and 3 first.

In general we would first allocate from the row with the largest penalty, then from the row with the second largest, and so on. Although the actual algorithm is much more complicated, using column as well as row penalties and using elimination of rows and columns with subsequent recomputation of penalties, the above example gives the basic idea.

The Vogel Approximation Method has been tested against full-blown transportation algorithms and has been found quite accurate for small matrices.

A.5 Missile Timing

The algorithm for determining the intersection of the timing line and the flight path for missiles with a LINE CORMSL uses the nature of the vector cross product to determine possible crossings.

Each great circle segment is the shorter great circle path between two points on the surface of the earth. By the nature of great circles, this path lies completely in the plane defined by the two end points

and the center of the earth. Given two such segments, the algorithm will calculate the point of intersection of the segments if they do cross.

In order to do this, we must define a three-dimensional Cartesian coordinate system and define a position vector.

We assume a right-handed coordinate system as shown in figure 136. The origin of the system is the center of the earth. The earth is assumed to have unit radius.

Define a position vector $r_i = (x_i, y_i, z_i)$ to be the vector originating at the origin and termination at some point on the earth's surface. Since this vector has unit length, we derive the following relationships between the end point's latitude and longitude and the Cartesian coordinates.

Define: β = latitude of end point (+ for North, - for South)
 α = longitude of end point, if East (360 - longitude, if West)

Then:

$$\vec{r}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} \cos \alpha \cos \beta \\ \sin \alpha \cos \beta \\ \sin \beta \end{pmatrix}$$

Therefore, each great circle can be defined by two position vectors.

Define: $\vec{R}_{ij} = \vec{r}_i \times \vec{r}_j$

R_{ij} is the cross product of two position vectors. This vector, R_{ij} , is perpendicular to the plane defined by the great circle. (See figure 137.) Any vector in that plane will be perpendicular to R_{ij} and any vector with base at the origin and perpendicular to R_{ij} will lie in the plane.

Define: \vec{r}_1 = position vector for first point on timing line
 \vec{r}_2 = position vector for second point on timing line
 \vec{r}_3 = position vector for launch point
 \vec{r}_4 = position vector for target.

\vec{R}_{12} is perpendicular to the plane of timing line
 \vec{R}_{34} is perpendicular to plane of flight path.

Let: $\vec{T} = \vec{R}_{12} \times \vec{R}_{34}$
 $= (\vec{r}_1 \times \vec{r}_2) \times (\vec{r}_3 \times \vec{r}_4)$

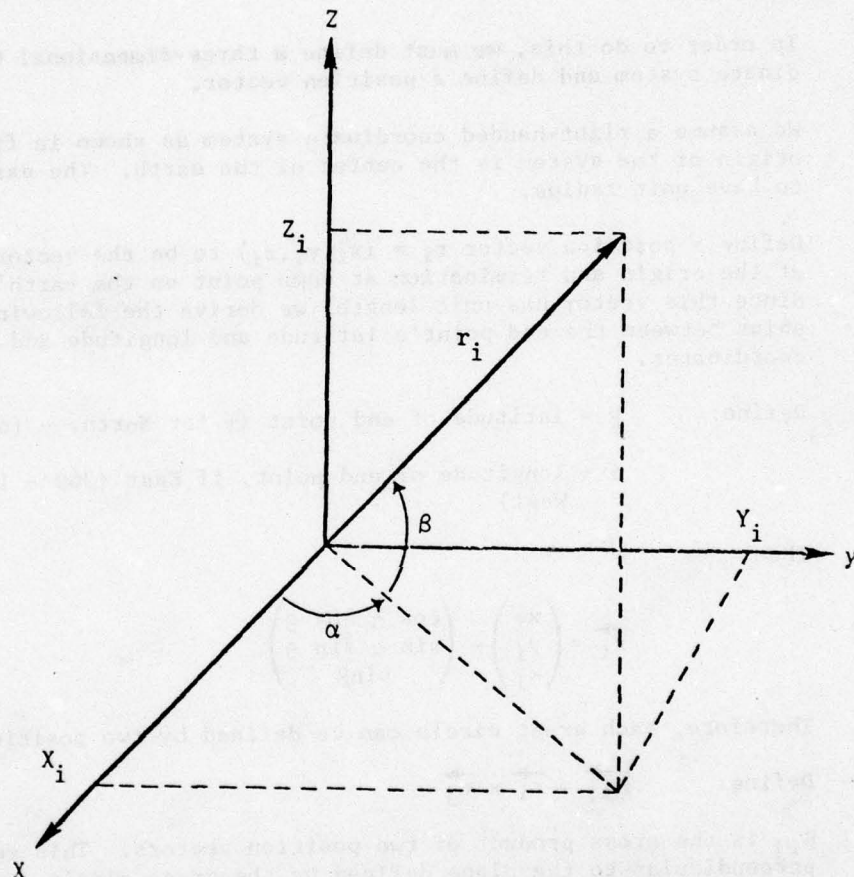


Figure 136. Coordinate System for Missile Timing Calculations

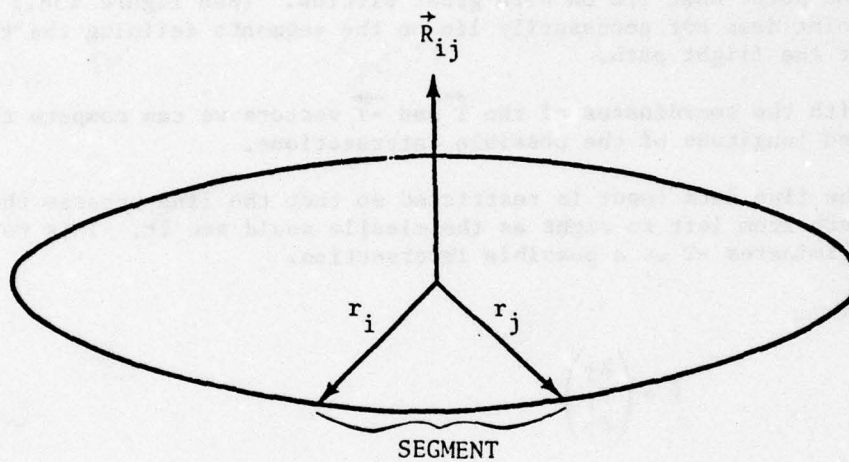


Figure 137. Relation of R_{ij} to Great Circle Plane

AD-A058 406

COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK). VOLU--ETC(U).
APR 78 D J SANDERS, P F MAYKRANTZ, J M HERRON

F/G 15/7

UNCLASSIFIED

CCTC-CSM-MM-9-77-VOL-4-PT SBIE-AD-E100 085

NL

5 OF 5
ADA
058406



END
DATE
FILMED
10 -78
DDC

If we normalize \vec{T} to have unit length, then \vec{T} and $-\vec{T}$ are position vectors. In fact, they are the position vectors for the points of intersection of the planes of the timing line and the flight path.

Since \vec{T} is perpendicular to \vec{R}_{12} , it lies in the first plane. Since it is perpendicular to \vec{R}_{34} , it lies in the second plane. Therefore, its end point must lie on both great circles. (See figure 138.) The end point does not necessarily lie on the segments defining the timing line or the flight path.

With the coordinates of the \vec{T} and $-\vec{T}$ vectors we can compute the latitude and longitude of the possible intersections.

The line data input is restricted so that the line crosses the flight path from left to right as the missile would see it. This restriction eliminates $-\vec{T}$ as a possible intersection.

If

$$\vec{T} = \begin{pmatrix} X_T \\ Y_T \\ Z_T \end{pmatrix}$$

then

$$\beta = \sin^{-1} (Z_T)$$

$$\alpha = \tan^{-1} (Y_T/X_T)$$

where the value of the arc tangent is not necessarily the principal value.

We now test these possible intersections to see if they lie on the segment as well as in the plane

Define: $D(\vec{r}_i, \vec{r}_j)$ = shorter great circle distance between end points of \vec{r}_i and \vec{r}_j

The possible intersection defined by \vec{T} lies on both segments if

$$D(\vec{r}_1, \vec{r}_2) = D(\vec{r}_1, \vec{T}) + D(\vec{r}_2, \vec{T})$$

and

$$D(\vec{r}_3, \vec{r}_4) = D(\vec{r}_3, \vec{T}) + D(\vec{r}_4, \vec{T})$$

If both these relations are true, then the point defined by \vec{T} is the intersection of the segments and that point is a crossing of the flight path and the timing line.

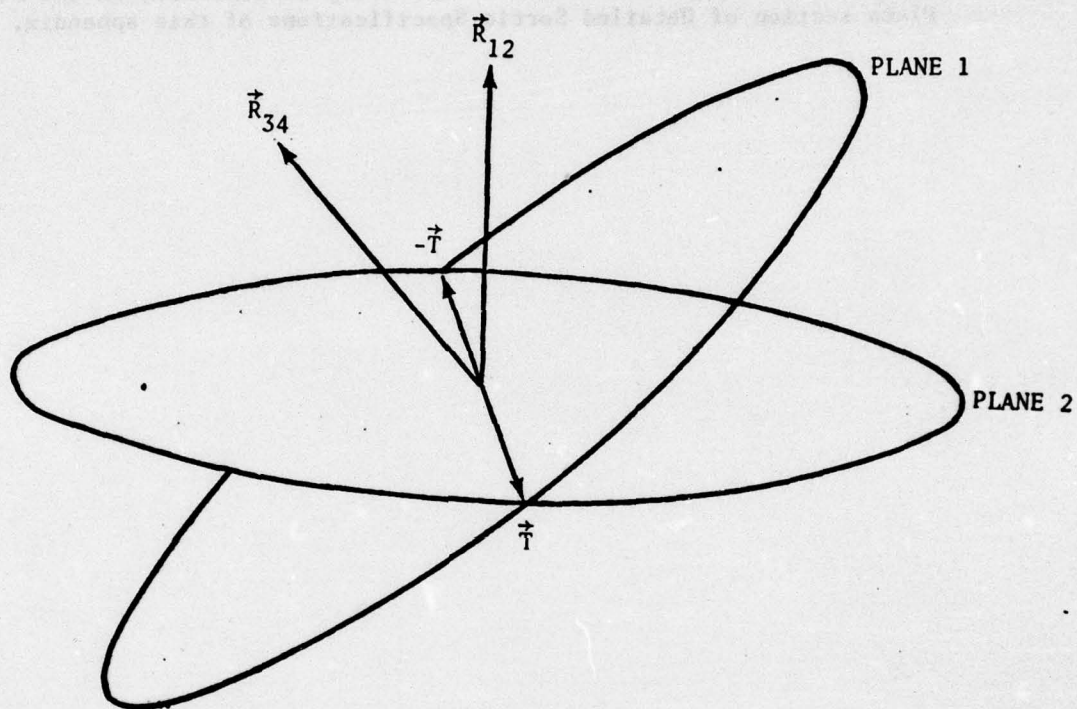


Figure 138. Diagram of T Vector

PLNTPLAN finds the time of the first crossing and uses that time to calculate the launch time so that the missile crosses the line at time equal to CORMSL plus the launch delay. If the missile does not cross any line, it will be launched to impact at game time equal to the launch delay. The calculation of the launch delay is described in the Missile Plans section of Detailed Sortie Specifications of this appendix.



APPENDIX B

AN ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

John D. C. Little
Massachusetts Institute of Technology

Katta G. Murty*
Indian Statistical Institute

Dura W. Sweeney**
International Business Machines Corporation

Caroline Karel
Case Institute of Technology

(Received March 6, 1963)

A 'branch and bound' algorithm is presented for solving the traveling salesman problem. The set of all tours (feasible solutions) is broken up into increasingly small subsets by a procedure called branching. For each subset a lower bound on the length of the tours therein is calculated. Eventually, a subset is found that contains a single tour whose length is less than or equal to some lower bound for every tour. The motivation of the branching and the calculation of the lower bounds are based on ideas frequently used in solving assignment problems. Computationally, the algorithm extends the size of problem that can reasonably be solved without using methods special to the particular problem.

The traveling salesman problem is easy to state: a salesman, starting in one city, wishes to visit each of $n-1$ other cities once and only once and return to the start. In what order should he visit the cities to minimize the total distance traveled? For 'distance' we can substitute time, cost, or other measure of effectiveness as desired. Distance or costs between all city pairs are presumed known.

The problem has become famous because it combines ease of statement with difficulty of solution. The difficulty is entirely computational, since a solution obviously exists. There are $(n-1)!$ possible tours, one or

* Work done while on a study assignment at Case Institute of Technology.

** Work done while a Sloan Fellow at M.I.T.

more of which must give minimum cost. (The minimum cost could conceivably be infinite--it is conventional to assign an infinite cost to travel between city pairs that have no direct connection.)

The traveling salesman problem recently achieved national prominence when a soap company used it as the basis of a promotional contest. Prizes up to \$10,000 were offered for identifying the most correct links in a particular 33-city problem. Quite a few people found the best tour. (The tie breaking contest for these successful mathematicians was to complete a statement of 25 words or less on "I like . . . because . . .".) A number of people, perhaps a little over-educated, wrote the company that the problem was impossible--an interesting misinterpretation of the state of the art.

For the early history of the problem, see Flood.¹ In recent years a number of methods for solving the problem have been put forward. Some suffer from inefficiency, others produce solutions that are not necessarily optimal, and still others require intuitive judgments that would be hard to program on a computer. For a detailed discussion, see Gonzalez.² We shall restrict our discussion to methods that (1) guarantee optimality, (2) seem reasonable to program, and (3) are general, i.e., not ad hoc to the specific numerical problem.

Among such methods the approach that has been carried furthest computationally is that of dynamic programming. Held and Karp³ and Gonzalez² have independently applied the method and have solved various test problems on computers. Gonzalez programmed an IBM 1620 to handle problems up to 10 cities. In his work the time to solve a problem grew somewhat faster than exponentially as the number of cities increased. A 5-city problem took 10 seconds, a 10-city problem took 8 minutes, and the addition of one more city multiplied the time by a factor, which, by 10 cities, had grown to 3. Storage requirements expanded with similar rapidity.

Held and Karp³ have solved problems up to 13 cities by dynamic programming using an IBM 7090. A 13-city problem required 17 seconds. But such is the power of an exponential that, if their computation grows at the same rate as that of Gonzalez, a 20-city problem would require about 10 hours. Storage requirements, however, may become prohibitive before then. For larger problems than 13 cities, Held and Karp develop an approximation that seems to work well but does not guarantee an optimal tour.

We have found two papers in which the problem has been approached by methods similar to our 'branch and bound' algorithm. Rossman, Twery, and Stone⁴ in an unpublished paper apply ideas that they have called combinatorial programming.⁵ To illustrate their method they present a 13-city problem. It was solved in 8 man-days. We have solved their

problem by hand in about $3\frac{1}{2}$ hours. Eastman,⁶ in an unpublished doctoral thesis and laboratory report, presents a method of solution and several variations on it. His work and ours contain strong similarities. However, to use our terminology, his ways of choosing branches and of calculating bounds are different from ours. He basically solves a sequence of assignment problems that give his bounds. We have a simpler method, and for branching we use a device with quite a different motivation. The biggest problem Eastman solves is 10 cities and he gives no computation times, so that effective comparisons are difficult to make.

Most published problems are symmetric, i.e., the distance from city i to city j is the same as from j to i . The algorithm to be presented also works for asymmetric problems; in fact, it seems to work better. Asymmetric problems arise in various applications. As an example from production scheduling, suppose that there is a production cycle of some time period, during which an assembly line must produce each of n different models. The cost of switching from model i to model j is C_{ij} . What order of producing models minimizes total setup cost? This is a traveling salesman problem in which it would not necessarily be expected that $C_{ij}=C_{ji}$.

To summarize, 13 cities is the largest problem which we know about that has been solved by a general method which guarantees optimality and which can reasonably be programmed for a computer. Our method appreciably increases this number. However, the time required increases at least exponentially with the number of cities and eventually, of course, becomes prohibitive. Detailed results are given below.

THE ALGORITHM

The basic method will be to break up the set of all tours into smaller and smaller subsets and to calculate for each of them a lower bound on the cost (length) of the best tour therein. The bounds guide the partitioning of the subsets and eventually identify an optimal tour--when a subset is found that contains a single tour whose cost is less than or equal to the lower bounds for all other subsets, that tour is optimal.

The subsets of tours are conveniently represented as the nodes of a tree and the process of partitioning as a branching of the tree. Hence we have called the method "branch and bound."

The algorithm will simultaneously be explained and illustrated by a numerical example. The explanation does not require reference to the example, however, for those readers who wish to skip it.

Notation

The costs of the traveling salesman problem form a matrix. Let the cities be indexed by $i=1, \dots, n$. The entry in row i and column j of the matrix is the cost for going from city i to city j . Let

$$C = [c(i,j)] = \text{cost matrix.}$$

C will start out as the original cost matrix of the problem but will undergo various transformations as the algorithm proceeds.

A tour, t , can be represented as a set of n ordered city pairs, e.g.,

$$t = [(i_1, i_2)(i_2, i_3) \dots (i_{n-1}, i_n)(i_n, i_1)],$$

which form a circuit going to each city once and only once. Each (i,j) represents an arc or leg of the trip. The cost of a tour, t , under a matrix, C , is the sum of the matrix elements picked out by t and will be denoted by $z(t)$:

$$z(t) = \sum_{(i,j) \text{ in } t} C(i,j).$$

Notice that t always picks out one and only one cost in each row and in each column. Also, let

- X, Y, \bar{Y} = nodes of the tree;
- $w(X)$ = a lower bound on the cost of the tours of X , i.e., $z(t) \geq w(X)$ for t a tour of X ;
- z_0 = the cost of the best tour found so far in the algorithm

Lower Bounds

A useful concept in constructing lower bounds will be that of reduction. If a constant, h , is subtracted from each element of a row of the cost matrix, the cost of any tour under the new matrix is h less than under the old. This is because every tour must contain one and only one element from that row. The relative costs of all tours are unchanged, however, and so any tour optimal under the old will be optimal under the new.

The process of subtracting the smallest element of a row from each element in the row will be called reducing the row. A matrix with non-negative elements and at least one zero in each row and column will be called a reduced matrix and may be obtained, for example, by reducing rows and columns. If $z(t)$ is the cost of a tour T under a matrix before reduction, $z_1(t)$ is the cost under the matrix afterward, and h the sum of constants used in making the reduction, then

$$z(t) = h + z_1(t). \quad (1)$$

Since a reduced matrix contains only nonnegative elements, h constitutes a lower bound on the cost of t under the old matrix.

Consider then the 6-city problem shown in figure 139. Reduction of the matrix by rows, then columns, give the matrix of figure 140. The total reduction is 48 so that $z(t) \geq 48$ for all t .

Branching

The splitting of the set of all tours into disjoint subsets will be represented by the branching of a tree, as illustrated in figure 141. The node containing 'all tours' is self-explanatory. The node containing i, j represents all tours which include the city pair (i, j) . The node containing i, j represents all tours to do not. At the i, j node there is another branching. The node containing k, l represents all tours that include (i, j) but not (k, l) , whereas k, l represents all tours that include both (i, j) and (k, l) . In general, by tracing from a node, X , back to the start, we can pick up which city pairs are committed to appear in the tours of X and which are forbidden from appearing. If the branching process is carried far enough, some node will eventually represent a single tour. Notice that at any stage of the process, the union of the sets represented by the terminal nodes is the set of all tours.

When a node X branches into two further nodes, the node with the newly committed city pair will frequently be called Y and the node with the newly forbidden city pair. Y .

		To					
		1	2	3	4	5	6
From	1	∞	27	43	16	30	26
	2	7	∞	18	1	30	25
	3	20	13	∞	35	5	0
	4	21	16	25	∞	18	18
	5	12	46	27	48	∞	5
	6	23	5	5	9	5	∞

Figure 139. Cost Matrix for a 6-city Problem. (A typical tour might be $t = [(1,3)(3,2)(2,5)(5,6)(6,4)(4,1)]$, which has the cost (length) $z = 43 + 13 + 30 + 5 + 9 + 21 = 121$.)

	1	2	3	4	5	6
1	∞	11	27	$\textcircled{10}$ 0	14	10
2	1	∞	15	$\textcircled{1}$ 0	29	24
3	15	13	∞	35	5	$\textcircled{5}$ 0
4	$\textcircled{1}$ 0	$\textcircled{0}$ 0	9	∞	2	2
5	2	41	22	43	∞	$\textcircled{2}$ 0
6	13	$\textcircled{0}$ 0	$\textcircled{9}$ 0	4	$\textcircled{2}$ 0	∞

Figure 140. Cost Matrix after Reducing Rows and Columns.
(Circled numbers are values of $\theta(i,j)$.)

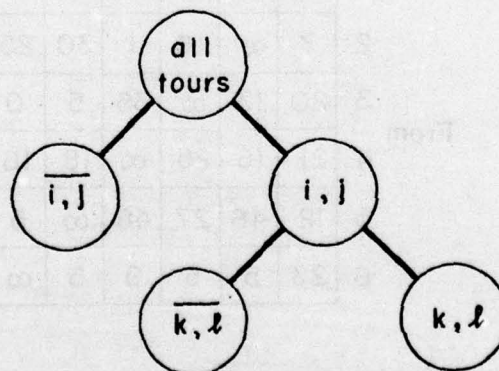


Figure 141. Start of Tree

Flow Chart

The workings of the algorithm will be explained by tracing through the flow chart of figure 142.

Box 1 starts the calculation by putting the original cost matrix of the problem into C , setting $X=1$ to represent the node, "all tours," and setting the cost of the best tour so far to infinity.

Box 2 reduces the matrix and labels node X with its lower bound $w(X)$.

Box 3 selects (k,l) , the city pair on which to base the next branching. The goal in doing this is to split the tours of X into a subset (Y) that is quite likely to include the best tour of the node and another (Y) that is quite unlikely to include it. Possible low cost tours to consider for Y are those involving an (i,j) for which $c(i,j)=0$.

Consider, therefore, the costs for tours that do not contain (i,j) , i.e., possible tours for Y . Since city i must be reached from some city, these tours must incur at least the cost of the smallest element in row i , excluding $c(i,j)$. Since city j must connect to some city, the tours must incur at least the cost of the smallest element in column j , excluding $c(i,j)$. Call the sum of these two costs $\theta(i,j)$. We shall choose (k,l) to be that city pair that gives the largest $\theta(i,j)$. (This amounts to a search over (i,j) such that $c(i,j)=0$, since otherwise $\theta(i,j)=0$.) Notice that, if $c(i,j)$ is set to infinity and then row i and column j are reduced, the sum of the reducing constants is $\theta(i,j)$.

For the example, the $\theta(i,j)$ values are written in small circles placed in the cells of the zeros of figure 140. The largest θ is $\theta(1,4)=10+10$ and so $(1,4)$ will be the first city pair used for branching.

Box 4 extends the tree from node X to \bar{Y} . As will be shown below, $w(\bar{Y})=w(X)+\theta(k,l)$. In the example $w(\bar{Y})=10+48=58$ and the node is so labeled in figure 143.

Box 5 sets up Y . Since the city pair (k,l) is now committed to the tours, row k and column l are no longer needed and are deleted from C . Next, notice that (k,l) will be part of some connected path generated by the city pairs that have been committed to the tours of Y . Suppose the path starts at city p and ends at city m . (Possibly $p=k$ or $m=l$ or both.) The connecting of m to p should be forbidden for it would create a subtour (a circuit with less than n cities) and no subtour can be part of a tour. Therefore, set $c(m,p)=\infty$.

After these modifications C can perhaps be reduced in the following places: row m , column p , any columns that had a zero in row k , and any rows that had a zero in column l . All other rows and columns contain some zero that cannot have been disturbed. Let h be the sum of

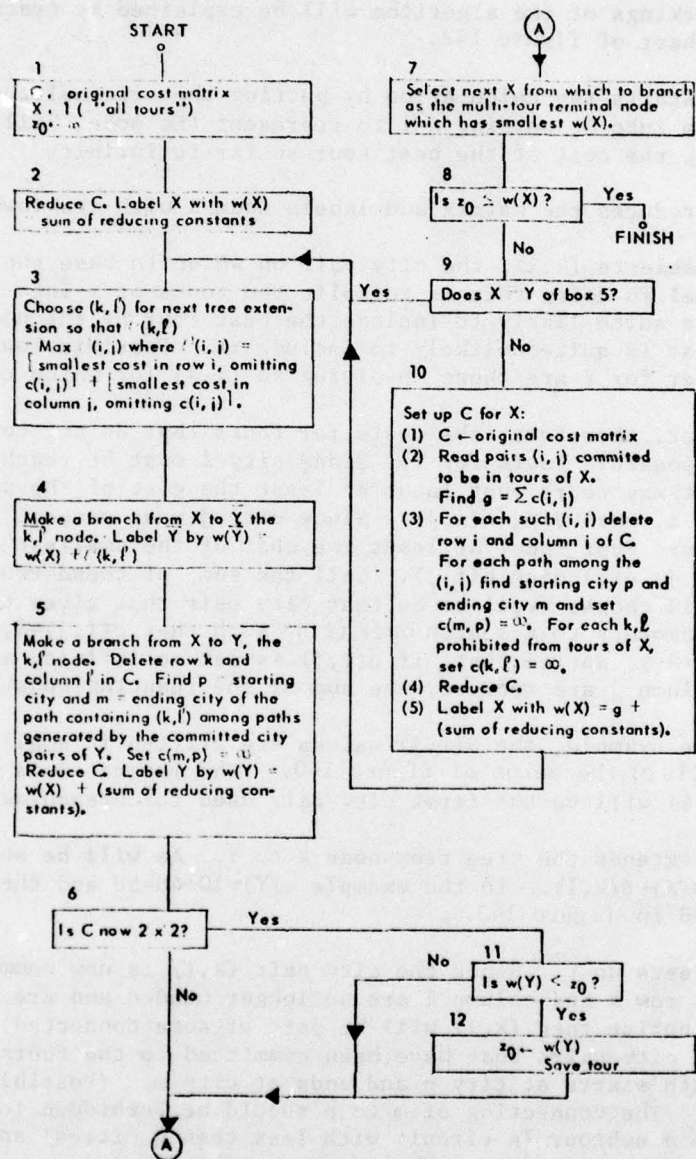


Figure 142. Flow Chart of the Algorithm

the new reducing constants. The lower bound for Y will now be shown to be

$$w(Y)=w(X)+h.$$

The algorithm operates so that the investigation of each node, X, starts in Box 3 with a matrix C and a lower bound $w(X)$ what stand in a special relation. If t is any tour of X, $z(t)$ its cost under the original matrix, t_1 , the city pairs of t left after removing those committed to the tours of X, and $z_1(t_1)$ the cost of t_1 under C, then it will be shown that

$$z(t)=w(X)+z_1(t_1). \quad (2)$$

This expression is true for the first node by (1). Suppose that from a bound $w(X_1)$ and matrix C_1 of a node X_1 , the algorithm constructs a bound $w(X_2)$ and reduced matrix C_2 for a node X_2 . (X_2 will be on some branch out of X_1 .) It will be shown that, if (2) is true for X_1 , (2) will also be true for X_2 .

The operations on C_1 to get C_2 (shown in Boxes 5 and 10) are always of the form: delete row i and column j for each (i,j) committed to the tours of X_2 , insert various infinities, reduce. The lower bound is always of the form

$$w(X_2)=w(X_1)+\sum c_1(i,j)+h, \quad (3)$$

where the summation is over the city pairs committed in X_2 but not in X_1 and h is the sum of the reducing constants. But consider any t in X_2 (and therefore in X_1). If we let $z_1(t_1)$ be the cost of the uncommitted city pairs of X_1 under C_1 and $z_2(t_2)$ be the cost of the uncommitted city pairs of X_1 under C_2 ,

$$z_1(t_1)=\sum c_1(i,j)+h+z_2(t_2),$$

or using (2), assumed true for X_1 ,

$$\begin{aligned} z(t) &= w(X_1) + \sum c_1(i,j) + h + z_2(t_2) \\ &= w(X_2) + z_2(t_2) \end{aligned}$$

so that (2) is true for X_2 , as was to be shown.

Equation (3) is used to calculate the lower bounds in Boxes 4, 5, and 10. The these lower bounds are valid is established by (2) and the non-negativity of the elements of C.

For the example, the matrix of Figure 143 shows the deletion of row 1 and column 4. The connected path containing (1,4) is (1,4) itself, so that $(m,p)=(4,1)$ and we set $c(4,1)=\infty$. Looking for reductions, we find that row 2 can be reduced by 1. Then $w(Y)=48+1=49$ as shown.

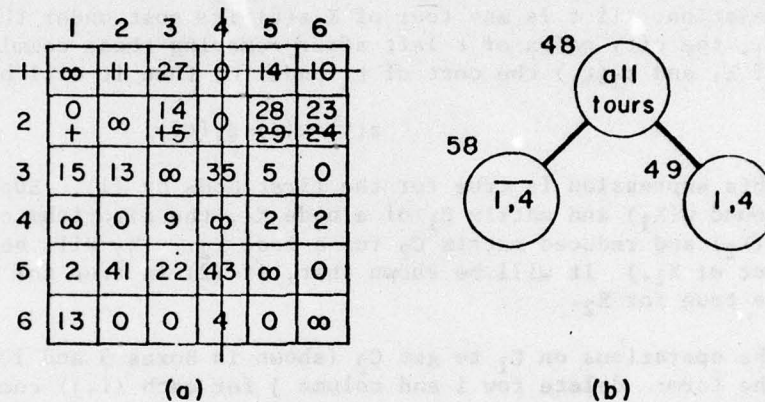


Figure 143. (a) Matrix after Deletion of Row 1 and Column 4
(b) First Branching

It is worth giving another example of finding (m,p) . Suppose the committed city pairs were (2,1), (1,4), (4,3), and (5,6) and $(k,1)$ were (1,4). Then the connected path containing $(k,1)$ would start at 2 and end at 3 to yield $(m,p)=(3,2)$.

Box 6 checks to see whether a single tour node is near.

Box 7 selects the next node for branching. There are a number of ways the choice might be made. The way shown here is to pick the node with the smallest lower bound. This leads to the fewest nodes in the tree.

Box 8 checks to see whether the algorithm is finished--whether the best tour so far has a cost less than or equal to the lower bounds on all terminal nodes of the tree.

Box 9 is a time saver. Most branching is from Y nodes, i.e., to the right. Such branching involves crossing out rows and columns and other manipulations that can be done on the matrix left over from the previous branching. When this case occurs, Box 9 detects it and the algorithm returns directly to Box 3.

Box 10 takes up the alternate case of setting up an appropriate lower bound and reduced matrix for an arbitrary X. Starting from the original cost matrix, rows and columns are deleted for city pairs committed to the tours of X, infinities are placed to block subtours and at forbidden city pairs, and the resulting matrix is reduced. The lower bound can be computed from (3) by thinking of X_1 in (3) as a starting node with $w(X_1)=0$ and matrix equal the original cost matrix. Since different ways of reducing a matrix may lead to different sums for the reducing constants, the recalculated $w(X)$ is substituted for the former one.

Boxes 11 and 12 finish up a single tour node. By the time C is 2×2 matrix, there are only two feasible (i,j) left and they complete a tour. Since the box is entered with a reduced matrix, the costs of the final commitments are zero, and $z=w(Y)$ by (2). If $z < z_0$, the new tour is the best yet and is read off the tree to be saved.

Returning to the example, Box 7 picks 1,4 as the second node for branching and, since this is a branching to the right, C is already available in reduced form. As shown in figure 144, the next branching is on the basis of (2,1) with $(m,p)=(4,2)$. Next, we go to the right from 2,1 on the basis of (5,6) with $(m,p)=(6,5)$ and then from 5,6 on the basis of (3,5) with $(m,p)=(6,3)$. At this point C is a 2×2 matrix, and we jump to Box 11 to finish the tour. We find $z=63$, which is stored as z_0 but, on returning to Boxes 7 and 8, we see that 1,4 has a lower bound of 58. To set up this node we go through Box 10. After the next branching, however, Box 8 shows that the problem is finished.

Discussion

At this point, let us stand back and review the general motivation of the algorithm. It proceeds by branching, crossing out a row and column, blocking a subtour, reducing the cost matrix to set a lower bound and then repeating. Although it is clear that the optimal solution will eventually be found, why should these particular steps be expected to be efficient? First of all, the reduction procedure is an efficient way of building up lower bounds and also of evoking likely city pairs to put into the tour. Branching is done so as to maximize the lower bound on the k,l node without worrying too much about the k,l node. The reasoning here is that the k,l node represents a smaller problem, once with the k th row and l th column crossed out. By putting the emphasis on a large lower bound for the larger problem, nonoptimal tours are ruled out faster.

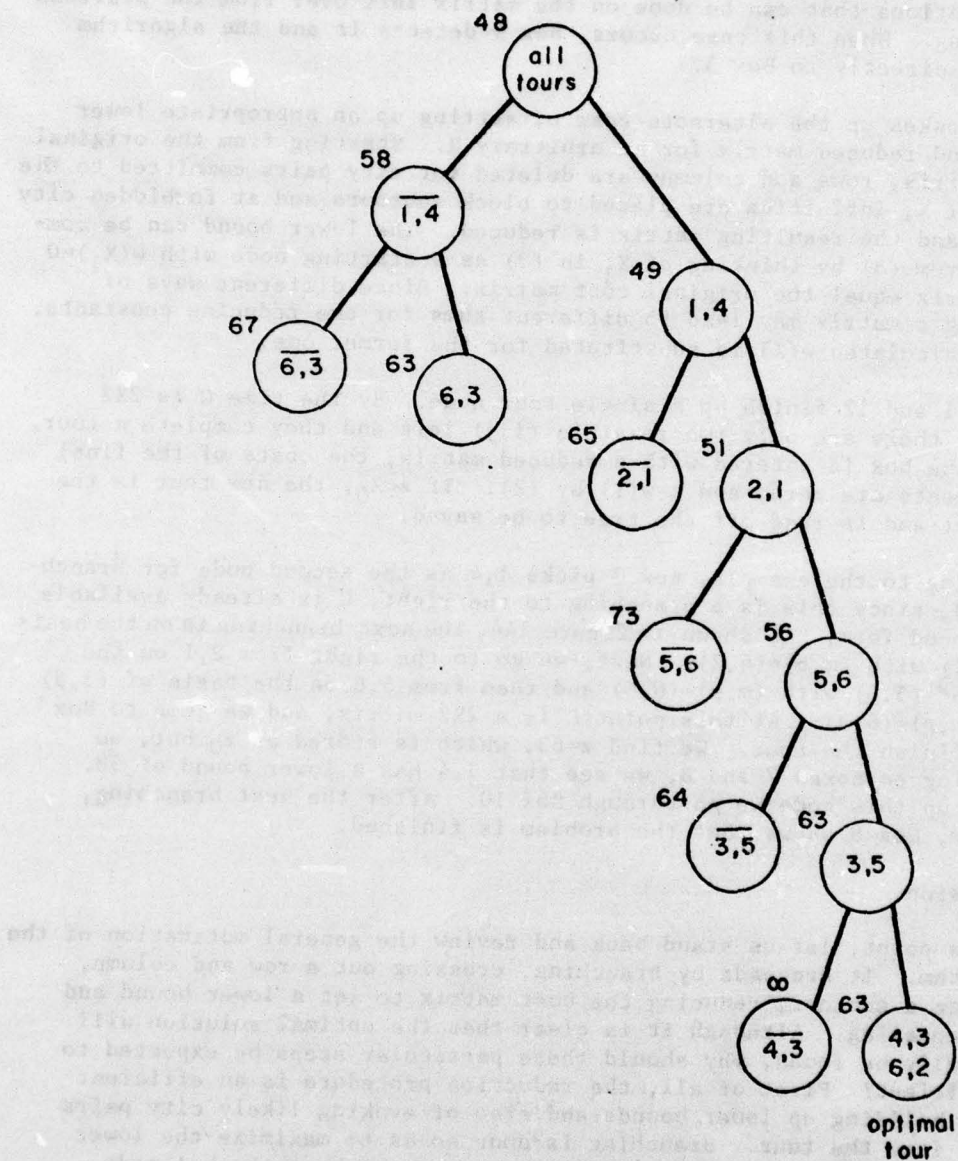


Figure 144. Final Tree

Insight into the operation of the algorithm is gained by observing that the crossing out of a row and column and the blocking of the corresponding subtour creates a new traveling salesman problem having one fewer city. Using the notation of Box 5, we can think of city m and city p as coalesced into a single city, say, m' . Setting $c(m,p)=\infty$ is the same as setting $c(m',m')=\infty$. The blocking of subtours is a way of introducing the tour restrictions into what is otherwise an assignment problem and is accomplished rather successfully by the algorithm.

Finally, unlike most mathematical programming algorithms, the one here has an extensive memory. It is not required that a trial solution at any stage be converted into a new and better trial solution at the next stage. A trial branch can be dropped for a moment while another branch is investigated. For this reason there is considerable room for experiment in how the next branch is chosen. On the other hand the same property leads to the ultimate demise of the computation--for n sufficiently large there are just too many branches to investigate and a small increase in n is likely to lead to a large number of new nodes that require investigation.

MODIFICATIONS

A variety of embellishments on the basic method can be proposed. We record several that are incorporated in the computer program used in later calculations. For the program listing itself (see Sweeney.⁹)

Go to the Right

It is computationally advantageous to keep branching to the right until it becomes obviously unwise. Specifically, the program always branches from the k,l node unless its lower bound exceeds or equals the cost of a known tour. As a result a few extra nodes may be examined, but usually there will be substantial reduction in the number of time-consuming setups of Box 10.

One consequence of the modification is that the calculation goes directly to a tour at the beginning. Then, if the calculations are stopped before optimality is proven, a good tour is available. There is also available a lower bound on the optimal tour. The bound may be valuable in deciding whether the tour is sufficiently good for some practical purpose.

Throw Away the Tree

A large problem may involve thousands of nodes and exceed the capacity of high speed storage. Storage can be saved, although usually at the expense of time, by noting that, at any point in the computation, the cost of the best tour so far sets an upper bound on the cost of an

optimal tour. Let the calculation proceed by branching to the right (storing each terminal node) until a single tour is found with some cost, say z_0 . Normally, one would next find the terminal node with the smallest lower bound and branch from there. Instead, work back through the terminal nodes, starting from the single tour, and discard nodes from storage until one is found with a lower bound less than z_0 . Then, branch again to the right all the way to a single tour or until the lower bound on some right-hand node builds to z_0 . (If the branch goes to the end, a better tour may be found and z_0 assigned a new, lower value.) Repeat the procedure: again work up the branch, discarding terminal nodes with bounds equal or greater than z_0 until the first once smaller is found; again branch to the right, etc.

The effect of the procedure is that very few nodes need to be kept in storage--something on the order of a few n . These form an orderly sequence stretching from the current operating node directly back to the terminal node on the leftmost branch out of "all tours."

As an illustration, consider the problem and tree of figure 146. The computation would proceed by laying out in storage the nodes $4,1$; $2,1$; $5,6$; and $3,5$. At the next step we find a tour with $z_0=63$ and the obviously useless node $4,3$. The tour is stored separately from the tree. Working up the branch, first $3,5$ is discarded, then $5,6$ and $2,1$, but $1,4$ has a bound less than z_0 . Therefore, branching begins again from there. A node $6,3$ is stored and then we find the node to the right has a bound equal(s) z_0 and may be discarded. Working back up the tree again, $6,3$ is discarded and, since that was the only remaining terminal node, we are finished.

The procedure saves storage but sometimes increases computation time. If the first run to the right turns up a rather poor tour, i.e., large z_0 , the criterion for throwing away nodes is too stiff. The calculation is forced to branch out from many nodes whose lower bounds actually exceed the cost of the optimal tour. The original method would never do this for it would never explore such nodes until it had finished exploring every node with a smaller bound. In the process, the optimal tour would be uncovered and so the nodes with larger bounds would never be examined.

Taking Advantage of Symmetry

If the traveling salesman problem is symmetric and t is any tour, another tour with the same cost is obtained by traversing the circuit in the reverse direction. Probably the most promising way to handle this is to treat the city pair (i,j) as not being ordered. This leads naturally to a new and somewhat more powerful bounding procedure. Although the basic ideas are not changed much, considerable reprogramming is required. So far, we have not done it.

There is another way to take advantage of symmetry and this one is easy to incorporate into our program. All reverse tours can be prohibited by modifying the nodes along the leftmost branch of the tree. These are the nodes with no city pairs committed but some forbidden. Suppose that such a node, X, branches into nodes, Y with (k,l) committed, and \bar{Y} , with (k,l) forbidden. The reverse tours of Y all have (l,k) in them. They cannot be in Y for the presence of both (k,l) and (l,k) is not possible in any tour. Such of the reverse tours as were in X are in \bar{Y} . We may prohibit them by setting $c(l,k)=\infty$ (as well as $c(k,l)=\infty$) in any matrix for Y. Thus, a reverse tour is prohibited as soon as the tour itself is identified to the extent of having one committed city pair.

A Computational Aid

In both hand and machine computation (k,l) is easiest calculated by first finding, for each row k and column l of the reduced matrix:

$\alpha(k)$ =the second smallest cost in row k.

$\beta(l)$ =the second smallest cost in column l.

Then $(k,l)=\alpha(k)+\beta(l)$ for any (k,l) which has $c(k,l)=0$. In a hand computation the $\alpha(k)$ can be written as an extra column to the right of the matrix and the $\beta(l)$ as an extra row at the bottom. After working out a few problems, one can see that when the branching is to the right there is no need to search the whole matrix to reset α and β , but that only certain rows and columns need be examined.

Other Possibilities

If desired, the algorithm can be modified so as to generate all optimal solutions. Instead of discarding nodes with $w(X)=z_0$, split them up further until eventually all the terminal nodes either have $w>z_0$ or are optimal single tours with $z=z_0$. Our computer program does not include this modification because in some cases it will increase the computing time a great deal--suppose the cost matrix were all zeroes.

Quite possibly, the average computing time can be decreased by solving the assignment problem for the original cost matrix and reducing the matrix by the cost of the optimal assignment in Box 2. (Some methods for solving the assignment problem leave it in reduced form.). The advantage lies in the larger lower bound with which the problem starts. The closer the starting lower bound to the cost of the optimal tour, the less is the branching that may be expected. Our exploration of the possible gains has not been extensive and has yielded mixed results: Croes' 20-city problem⁸ was speeded up, but some others were lengthened.

The idea that we are calling 'branch and bound' is more general than the traveling salesman algorithm. A minimal solution for a problem can

be found by taking the set of all feasible solutions, splitting it up into disjoint subsets, finding lower bounds on the objective function for each subset, splitting again the subset with the smallest lower bound, and so forth, until an optimal solution is found. The efficiency of the process, however, rests very strongly on the devices used to split the subsets and to find the lower bounds. As a simple example of another use of the method, if the step of setting $c(m,p)=\infty$ is omitted from the traveling salesman algorithm, it solves the assignment problem. For another example, see Doig and Land.¹⁰

Table 15. Mean and Standard Deviation of T for Random Distance Matrices
(T=time in minutes used to solve Traveling Salesman Problem on IBM 7090)

Number of cities	Number of problems solved	Mean T	Std. dev. T	Mean ^(a) log T	Std. dev. ^(a) log T
10	100	0.012	0.007	log 0.015	log 1.24
20	100	0.084	0.063	log 0.067	log 2.00
30	100	0.975	1.240	log 0.63	log 2.04
40	5	8.37	10.2	log 4.55	log 3.74

^(a) Obtained by plotting the cumulative frequency on log normal probability paper and fitting a straight line, except in the 40 cities case for which the computation was numerical. In the case of 10 cities the log normal fits only the tail of the distribution—30 per cent of the problems went directly to the solution without extra branching and thereby produced a lump of probability at 0.002 minute.

ACKNOWLEDGMENT

The computing time used in the early stages of the work was provided by the Case Institute of Technology Computing Center, the M.I.T. Computation Center, and the M.I.T. School of Industrial Management Computing Facility. The time for the production runs was provided by I.B.M. on the IBM 7090 at the M.I.T. Computation Center. We wish to acknowledge the help of R.H. Gonzalez for programming certain investigations. His time was supported by the U.S. Army Research Office (Durham).

DISTRIBUTION

<u>Addressee</u>	<u>Copies</u>
CCTC Codes	
Technical Library (C124)	3
C124 (Stock)	6
C313	1
C314	17
C600	1
DCA Code	
205	1
EXTERNAL	
Chief, Studies, Analysis and Gaming Agency, OJCS ATTN: SFD, Room 1D957, Pentagon, Washington, DC 20301	2
Chief of Naval Operations, ATTN: OP-96C4, Room 4A478, Pentagon, Washington, DC 20350	2
Commander-in-Chief, North American Air Defense Command ATTN: NPXYA, Ent Air Force Base, CO 80912	2
Commander, U. S. Air Force Weapons Laboratory (AFSC) ATTN: AFWL/SUL (Technical Library), Kirtland Air Force Base, NM 87117	1
Director, Strategic Target Planning, ATTN: (JPS), Offutt Air Force Base, NE 68113	2
Defense Documentation Center, Cameron Station, Alexandria, VA 22314	12
	50

DISTRIBUTION

<u>Addressee</u>	<u>Copies</u>
CCTC Codes	
Technical Library (C124)	3
C124 (Stock)	6
C313	1
C314	17
C600	1
DCA Code	
205	1
EXTERNAL	
Chief, Studies, Analysis and Gaming Agency, OJCS ATTN: SFD, Room 1D957, Pentagon, Washington, DC 20301	2
Chief of Naval Operations, ATTN: OP-96C4, Room 4A478, Pentagon, Washington, DC 20350	2
Commander-in-Chief, North American Air Defense Command ATTN: NPXYA, Ent Air Force Base, CO 80912	2
Commander, U. S. Air Force Weapons Laboratory (AFSC) ATTN: AFWL/SUL (Technical Library), Kirtland Air Force Base, NM 87117	1
Director, Strategic Target Planning, ATTN: (JPS), Offutt Air Force Base, NE 68113	2
Defense Documentation Center, Cameron Station, Alexandria, VA 22314	12
	50

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSM MM 9-77, Volume IV, Parts I&II	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK), Program Maintenance Manual, Sortie Gen- eration Subsystem		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Dale J. Sanders Paul F. M. Maykrantz Jim M. Herron		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS System Sciences, Incorporated 4720 Montgomery Lane Bethesda, Maryland 20014		8. CONTRACT OR GRANT NUMBER(s) DCA 100-75-C-0019
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center Room BE-685, The Pentagon, Washington, DC 20301		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 15 April 1978
		13. NUMBER OF PAGES 680
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) War Gaming, Resource Allocation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide statistical output summaries, and produce input tapes to simulator subsystems external to QUICK. The Program Maintenance Manual consists of four volumes which facilitate maintenance of the war gaming system. This volume, Volume IV, provides the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the modules and subroutines of		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
661

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PRECEDING PAGE BLANK-NOT FILLED

CONT

20. (Continued)
the Sentic Generation Subsystem.

The Program Maintenance Manual complements the other QUICK Manuals to facilitate application of the war gaming system. These manuals Series 9-77 are published by the Command and Control Technical Center (CCTC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.